uc3m | Universidad Carlos III de Madrid

Grado de Ingeniería Informática
2017-18

Trabajo Fin de Grado

Desarrollo de Heurísticas en Planificación Automática mediante PDBs

---

Autor
José González Barroso

Tutor
Daniel Borrajo Millán

Campus: **Leganés**

En un mundo donde la ausencia de la ignorancia es una falacia,
el único que impide el progreso humano es el necio.
Donde muere el raciocinio, triunfa la tiranía.
- José González Barroso

AGRADECIMIENTOS

Quiero agradecer, en primer lugar, a mis padres y mi hermano, por apoyarme durante estos 4 años duros de carrera. Personas que siempre han confiado en mí y me han inculcado valores como la constancia, el trabajo duro y bien hecho y el esfuerzo. Bajo el lema de que todo se recompensará, han sabido guiarme por el mejor camino.

También quiero agradecer a la universidad Carlos III por formarme durante estos años, no solo a nivel profesional, sino también a nivel personal. No me equivocaría en afirmar que nosotros, los estudiantes, adquirimos unas competencias que muchas otras universidades desearían aportar. Por supuesto, deseo darle las gracias a todos los profesores que me han hecho llegar hasta donde estoy, no solo en la universidad, sino también en el colegio y en el instituto. En especial a cinco de ellos:

Primero, a mi maestro de quinto y sexto de primaria, don Vicente, que supo inculcarme ciertos valores que actualmente sigo aplicando. Muchas veces olvidamos la importancia de los maestros en el colegio y no los tenemos en cuenta. Pero, nada más lejos de la realidad, un buen desempeño en su trabajo influye, y mucho, en la ilusión de los alumnos de seguir formándose. Si he llegado a donde estoy gran parte es gracias a él. En segundo lugar, a Salvador Campoy, profesor de matemáticas en mi antiguo instituto, que me ayudó a ilusionarme y amar las matemáticas, las ciencias puras, la teoría y la investigación. También, agradezco a Félix García Carballeira por ayudarme a no abandonar la carrera, y es que, aunque algunos profesores lo desconozcan, muchos de ellos consiguen que volvamos a interesarnos por materias en las que habíamos perdido la ilusión. Agradezco a mi profesor de cálculo diferencial, Manuel Carretero Cerrajero, por su gran habilidad en la enseñanza de las matemáticas. Y es que, gracias a él, se convirtió en mi asignatura favorita. Y, por último, quiero agradecer a mi tutor Daniel Borrajo Millán por animarme a realizar este trabajo de investigación, por confiar en mí y por sacrificar parte de su tiempo en ayudarme cuando no sabía cómo avanzar. Me siento orgulloso de que él me haya iniciado en mi primer proyecto de investigación.

Estoy satisfecho de mi desempeño en la universidad. Como futuro graduado en ingeniería informática, estoy deseoso de aportar nuevo conocimiento a esta ciencia con mis trabajos e investigaciones en los próximos años que me deparan.

RESUMEN

La **planificación automática** es una disciplina utilizada hoy en día. Es por eso por lo que es objeto de **investigación**. Este estudio nace a raíz de querar mejorar los resultados obtenidos mediante las bases de datos de patrones, más conocidas con el nombre de **PDBs**. Esta mejoría se traduce en el desarrollo de una heurística más informada, en la menor utilización de recursos temporales y de memoria, y/o en la mayor cantidad de problemas resueltos por el planificador. El estudio se divide en dos partes: generación de heurísticas dependientes del dominio y en la generación de heurísticas independientes del dominio.

En el estudio **dependiente del dominio** se han seleccionado 5 dominios de **IPC** (*Snake*, *Termes*, *Hiking*, *Spider* y *Tetris*) y 3 algoritmos de cálculos de heurísticas de ***Fast Downward*** (*iPDB*, *Canonical PDB* y *Zero-One PDB*) con los que probar. Tras ejecutarlos, se han observado las características comunes entre las PDBs resultantes y se ha implementado un algoritmo que las crea directamente sin hacer uso de búsqueda. La conclusión alcanzada en esta primera parte del estudio es que realizar previamente un **análisis de las variables** y de cómo se agrupan formando PDBs al ejecutar ciertos algoritmos es un gran paso en el camino de la obtención de una buena heurística.

Tras esto, en el estudio **independiente del dominio** se han observado las características que existen en común entre todas las variables presentes en estas PDBs implementadas en el estudio dependiente del dominio. Se llegó a la conclusión de que coincidían en gran medida con los predicados presentes en las precondiciones de las acciones que originan un cambio en un predicado meta, teniendo en cuenta el signo del predicado meta (*not*), realizado una unión entre todos los conjuntos de predicados por cada una de las metas, y terminando con una intersección entre todos estos conjuntos. Posteriormente, se decidió modificar iPDB aumentando el *improvement* de las PDBs que contenían variables con estos predicados obtenidos tras la operación entre conjuntos. Las conclusiones obtenidas en este segundo estudio indican que la mejoría en iPDB no es tan notable, pero la suficiente para percatarse de que, mediante más investigaciones de este estilo, se podría llegar a unos muy buenos resultados.

Este estudio podría utilizarse, por tanto, como punto de partida de futuros proyectos de investigación acerca de las PDBs. Los buenos resultados obtenidos en este primer estudio base corroboran que más procesos de investigación en esta misma línea podrían dar lugar a unos mejores resultados.

Palabras clave: investigación, planificación automática, PDBs, *Fast Downward*, *iPDB*, IPC, heurísticas dependientes del dominio, heurísticas independientes del dominio, análisis de variables en PDBs.

ÍNDICE

ÍNDICE DE TABLAS

ÍNDICE DE ILUSTRACIONES

SECCIÓN 1. INTRODUCCIÓN Y OBJETIVOS

Además de la introducción y los objetivos, en este punto se abordará la estructura del documento, las convenciones que se siguen y las definiciones y las abreviaturas.

1.1 Introducción

El avance tecnológico durante esta última década está siendo abismal. Dispositivos, tecnologías, algoritmos y métodos que triunfaban hace unos años hoy en día están obsoletos, y no es de extrañar, pues estamos viviendo un gran *boom* en la digitalización. Hemos hecho de los dispositivos móviles una herramienta fundamental en nuestras vidas, y todo parece indicar que en el futuro todo irá a más.

Para llegar hasta aquí, han sido muchos aquellos que han invertido horas en investigar acerca de nuevos algoritmos y nuevas herramientas para mejorar nuestra calidad de vida. Para facilitarnos el trabajo. A muy pocos les extrañará saber que nuestros dispositivos móviles son más potentes que la tecnología que usaba la NASA en el año 1969. Aquella tecnología que utilizaron para llegar a la luna. Todo esto gracias a las investigaciones que se han realizado, que actualmente se están llevando a cabo y las que faltan por iniciar.

Como estudiante de ingeniería informática, lo que más me atrae de esta ciencia son las bases teóricas. Poder aportar mi granito de arena para mejorar la tecnología actual. La informática no solo sirve para realizar aplicaciones o programas comerciales. No es lo mismo ser un programador que un ingeniero, al igual que no es lo mismo un albañil que un arquitecto, sin desprestigiar a ninguna de las dos profesiones.

Personalmente, he tenido la oportunidad de realizar un TFG acerca de un trabajo de investigación. Concretamente en el área de la planificación automática. Para ello, se ha hecho uso del planificador *Fast Downward* (Helmert, 2018) y de algunos dominios y problemas de IPC (Keller, Sanner, & Say, 2018), *The International Planning Competition*. La idea inicial fue desarrollar una heurística que operase en un dominio concreto mediante PDBs, es decir, Base de Datos de Patrones; y ser capaz, posteriormente, de generalizar implementando una heurística independiente del dominio a partir del estudio realizado sobre estos dominios de IPC. Para ello, hay que analizar cuáles son las variables y los patrones implicados en la resolución de cada uno de estos dominios, resolviéndolos previamente con un planificador que hiciese uso de las PDBs. Una vez hecho esto, intentar generalizar para poder identificar automáticamente cuáles serían aquellas variables que podrían ser de utilidad en un dominio dado, y conseguir resolver la mayor parte de los problemas de dicho dominio en el menor tiempo y capacidad de memoria posibles.

Fast Downward proporciona algoritmos capaces de generar PDBs y resolver problemas, pero realizan previamente una búsqueda para ello. Estas búsquedas suelen acarrear un gasto de memoria y de tiempo considerables. La actual investigación nació a raíz de querer obtener un conjunto de patrones tan bueno o mejor que el que proporciona este tipo de algoritmos de búsqueda, pero de una forma más guiada y rápida. Sin embargo, el estudio es limitado, y se centra exclusivamente en la obtención de planes óptimos, no meramente satisfacibles. Las futuras investigaciones que se proponen en este documento abarcan más temas de estudio que podrían servir para mejorar en el ámbito de la planificación automática.

De esta forma, si alguien deseara obtener un plan óptimo a partir de un estado inicial para llegar a un *estado final* (estado en el que se cumple cada una de las metas del problema) mediante un conjunto de acciones, si usar PDBs está entre sus ideas, la información obtenida en este proyecto le podrá ser de gran utilidad.

1.2 Motivación y objetivos

La razón de ser de esta investigación, tal y como se ha redactado en la introducción, es ser capaz de resolver problemas de planificación en el menor tiempo y cantidad de memoria posibles, mediante el estudio de las variables y patrones implicados en la resolución de determinados dominios, para, de esta forma, poder generalizar sin necesidad de estudiar el dominio en cuestión. A continuación, se enumeran dichos objetivos de manera más formal y completa:

1. Realizar un estudio del estado del arte de acorde con los propósitos de este proyecto.
2. Abordar el marco regulador y aplicarlo al proyecto.
3. Planificar el proyecto, estableciendo un presupuesto inicial y el impacto socio-económico que este tiene.
4. Redactar requisitos, los casos de uso y todos estos aspectos relacionados con el diseño de la solución.
5. Identificar cuáles son las variables implicadas en la resolución de los problemas de planificación, y comprender cómo agruparlas en las PDBs.
6. Implementar un algoritmo que, dado un dominio y un conjunto de problemas de optimización, construya directamente las PDBs, probándolo, evaluándolo y aceptándolo si llega a ser tan bueno o mejor que el mejor algoritmo analizado de *Fast Downward* que los resuelva. Sacar conclusiones a raíz de dichos resultados. Realizar este estudio con 5 dominios: *Snake*, *Termes*, *Hiking*, *Spider* y *Tetris*.
7. A partir de la información extraída en cada uno de estos dominios, realizar un estudio para ser capaz de obtener estas variables y PDBs relevantes en la

resolución de un problema de planificación dado un dominio y un problema cualquiera, sin tener que estudiarlo y analizarlo previamente.
8. Implementar un algoritmo a partir de la información extraída, probarlo, evaluarlo y sacar conclusiones.
9. Plantear líneas de estudio futuras relacionados con el actual proyecto y sus resultados.
10. Redactar una memoria que recoja toda esta información (añadiendo la planificación y el presupuesto finales) y la exponga de manera clara, limpia y entendible; cumpliendo con todos los criterios de la matriz de corrección.

1.3 Estructura del documento

En este punto se comenta la estructura del documento. Este se rige por las matrices de corrección de los TFGs en la UC3M (UC3M, Matriz de Evaluación de Trabajo Fin de Grado, 2018) (UC3M, Informe del Tutor del Trabajo Fin de Grado, 2018) y por las pautas y recomendaciones de un documento que muestra cómo estructurar un TFG de ingeniería informática (UPV). Además, como guía para definir la estructura, se ha observado un TFG ya presentado (Maganto, 2017). Todos estos archivos se disponen en las referencias.

El actual documento se divide en 10 secciones:

1) La **introducción**.
    - La introducción da pie al contenido del TFG, es decir, situar al lector en el contexto en el que se desarrolla en proyecto.
    - Tras esto, se exponen en un siguiente punto la motivación y los objetivos.
    - Posteriormente se presenta la estructura del documento.
    - Se establecen las convenciones del documento, como el tamaño de letra, tipografía, espaciado, etc.
    - Finaliza con las definiciones y las abreviaturas que aparecen en el proyecto.
2) El **estado del arte**. El estado del arte habla acerca de toda la tecnología y la teoría que envuelve al TFG. Estas son:
    - La planificación automática.
    - Los algoritmos de búsqueda, centrándose en A* y *Hill climbing*.
    - El lenguaje PDDL.
    - La teoría detrás de las PDBs.
    - La herramienta *Fast Downward* junto a la generación de los archivos *output.sas* y tres de sus algoritmos de cálculo de heurísticas (*iPDB*, *Canonical PDB* y *Zero-One PDB*).

- El significado de IPC y cinco de sus dominios (*Snake*, *Termes*, *Hiking*, *Spider* y *Tetris*).
- Trabajos relacionados con el actual proyecto, seguido de la comparación entre estos.
- Una crítica al estado del arte.
- La propuesta que va a dar sentido al estudio llevado a cabo, y que pretende solventar los puntos abarcados en la crítica al estado del arte.

3) El **diseño de la solución**. Este punto se compone de diversos apartados. Se comentan aquellos que se relacionan con su estructura general:
   - La metodología completa del estudio, desde el establecimiento del esqueleto de la investigación hasta el establecimiento de trabajos futuros.
   - Las alternativas en la metodología del estudio, desde cambios livianos e insignificantes hasta modificaciones que podrían plantearse como idea para un potencial proyecto.
   - Cuestiones acerca del código implementado. En este punto se establecen algunas cuestiones generales, la plantilla de requisitos y se diseña la solución para cada uno de los 5 dominios y para el estudio independiente del dominio, finalizando con algunas aclaraciones acerca de *scripts* adicionales que se han implementado para agilizar el trabajo. Por cada uno de los dominios y para el estudio independiente del dominio se establecen los requisitos (de usuario, funcionales y no funcionales), los casos de uso, el manual de usuario, la arquitectura del sistema y las funciones implementadas.
4) La i**nvestigación**. Se podría dividir esta sección en tres puntos:
   - Un estudio previo donde se seleccionan los dominios y los algoritmos que se van a utilizar en el proyecto.
   - Por cada uno de los 5 dominios se realiza un proceso de investigación que consiste en ejecutar los algoritmos, observar sus resultados, implementar uno en función de estos resultados, ejecutarlo, hacer pruebas con él, evaluarlo con el resto de los algoritmos y extraer conclusiones.
   - El estudio independiente del dominio tiene una estructura similar: se observan las características similares entre las PDBs relevantes de cada dominio, se implementa uno en función de estas observaciones y/o se modifican otros, se ejecuta el nuevo algoritmo obtenido, se hacen pruebas con él, se evalúa comparando los resultados por cada dominio y se extraen conclusiones.
5) El **marco regulador**. En esta sección se abarcan tres puntos fundamentales:
   - Un análisis de la legislación aplicable al proyecto, analizando los riesgos que se suponen al apostar por la realización de este estudio, las responsabilidades profesionales y éticas, los riesgos laborales y las cuestiones acerca de la privacidad y la seguridad que se relacionan con

este. Además, se establecen los rangos salariales para poder realizar posteriormente el presupuesto.
    - Un análisis de los estándares técnicos acerca del producto final, la metodología llevada a cabo y las cuestiones acerca de la licencia, los términos y las condiciones de las herramientas *hardware*, *software* y lenguajes utilizados.
    - Se abordan, en último lugar, las cuestiones acerca de la propiedad intelectual.
6) El **entorno socio-económico**. Esta sección se desglosa en tres puntos:
    - La planificación del proyecto, es decir, la planificación inicial, la final y las desviaciones.
    - El presupuesto del proyecto, analizando los costes salariales, los dispositivos *hardware*, los complementos, las herramientas *software*, el material fungible, los viajes, las dietas y los costes indirectos. Tras este desglosamiento se establecerá un prespuesto inicial, presente en la oferta, y un coste final.
    - El impacto socio-ecnonómico que acarrea el proyecto.
7) Las **conclusiones generales**. En esta sección se establecen las conclusiones generales del proyecto, comentando también los siguientes puntos:
    - Dificultades personales y errores cometidos.
    - Relación del trabajo con los estudios cursados.
    - Valores y conocimiento adquiridos durante la realización del proyecto.
8) Los **trabajos futuros**. Este proyecto ha originado múltiples ideas para trabajos futuros, expuestas todas en esta sección.
9) El **anexo**. En este punto se incorpora toda la información adicional del proyecto que no se ha añadido en los puntos anteriores para no saturarlos. Estos son:
    - Abstract (resumen en inglés).
    - Algunas pruebas de los algoritmos implementados.
    - Algunas tablas con los resultados de la investigación, resultado del proceso de evaluación de los algoritmos implementados. Esas tablas contienen también la información del proceso de observación.
10) Las **referencias**. Todos los trabajos que se han utilizado en el proyecto se indican aquí.

1.4 Convenciones

En este punto se establece la normativa de marcado que va a seguir el documento **a grandes rasgos**, como el tamaño de los márgenes, el uso de la cursiva, etc. Se desarrollará en forma de lista:

- Los **márgenes** son: 2,5 cm superior e inferior y 3 cm laterales. Sin embargo, estos pueden romperse en caso de tener que introducir alguna tabla, gráfico o imagen de dimensiones superiores, teniendo que introducir una página en horizontal en caso necesario.

- El texto está **justificado**, excepto en las tablas y ecuaciones, donde puede aparecer centrado o alineado a la izquierda.

- La tipografía de todo el documento es ***calibri***, usando el tamaño de letra 12 de froma general.

- Los **títulos** de las secciones irán en negrita, acompañados de su número de índice. En el tercer nivel del índice irán solo subrayados y a partir del cuarto nivel tan solo con el número del índice y con sangría. En el primer nivel la letra tendrá tamaño 16 y en el segundo nivel, 14. En el resto de los niveles la letra tendrá tamaño 12.

- Los **títulos** de nivel uno se situan al inicio de una página. El resto de los títulos irán a continuación de la información previa, dejando un **salto de línea** de diferencia en caso de necesitarlo.

- La **negrita** también se utiliza para remarcar aspectos importantes del documento, con el fin de agilizar la lectura.

- El **subrayado** también se utiliza para remarcar aspectos importantes del documento cuando se quiere hacer algún tipo de distinción con las palabras en negrita, con el fin de agilizar la lectura.

- Los extranjerismos, algunos nombres propios y los términos acuñados en este documento irán en **cursiva**, exceptuando las siglas. Por ejemplo: *improvement*, *Ley de Propiedad Intelectual* o predicado *relevante*.

- El **espaciado** entre líneas y párrafos es de 1, excepto en las listas y con código fuente, donde se reduce ligeramente según convenga.

- Se hará uso de **listas numeradas** en caso de establecer un orden entre los elementos, aunque estas pueden ir también sin enumerar.

- Se hará uso de **listas sin enumerar** constantemente a lo largo del documento. En lugar de optar por una redacción extensa y tediosa, optar por listas sin enumerar es una gran alternativa para agilizar la lectura.

- El contenido de las tablas está en **tamaño** 11. Los títulos de las tablas, ilustraciones y ecuaciones irán en cursiva y con tamaño 9, en la parte inferior de estas y centrado.

- Las **citas texutales** irán entrecomilladas y las **referencias** en formato *APA*.

1.5 Definiciones y abreviaturas

Las *tablas 1 y 2* muestran las definiciones y las abreviaturas.

1.5.1 Definiciones

Término	Definición
Algoritmo	Conjunto de operaciones que pretende dar respuesta a cualquier problema relacionado con el objetivo de dicho algoritmo.
Altura variable	En el dominio *Termes*, son todas las variables con alturas que difieren del estado inicial respecto a un *estado final*.
Dominio	Conjunto de tipos, funciones, predicados y acciones que definen un *mundo* en planificación automática.
Estado final	Estado en el que se cumple cada una de las metas del problema a resolver.
Heurística	Valor numérico que indica la lejanía de un estado respecto a la meta.
Improvement	En *iPDB* (algoritmo de *Fast Downward*) es un valor numérico que refleja la calidad de una PDB respecto a las anteriores PDBs.
Meta variable	Equivalente al término *altura variable.*
Plan óptimo	Conjunto de acciones ordenadas que dan solución a un problema, usando el menor coste posible.
Plan satisfacible	Conjunto de acciones ordenadas que dan solución a un problema, sin necesidad de usar el menor coste posible.
Planificación automática	Disciplina dentro de la inteligencia artificial que se basa en la construcción de planes (conjunto de acciones) que, partiendo desde un estado inicial, desembocan en un estado que cumple cada una de las condiciones meta.
Predicado	Sentencia presente en la definición de los dominios que sirve para definir estados, mediante la instanciación de estos junto a los objetos definidos en el problema a resolver.

Predicado *relevante*	Predicado utilizado en el estudio independiente y/o dependiente del dominio.
Problema	Conjunto de instancias de tipos, un estado inicial y un conjunto de metas que definen una situación del dominio asociado.
Signo	Existencia o no de la sentencia *not* en la instanciación de un predicado, indicando si es real en el estado asociado.
Tipo	Sentencia presente en la especificación de los dominios que sirve para definir objetos en el problema.
Variable	Sentencia presente en el archivo *output.sas* que contiene un conjunto de instancias de predicados que no pueden darse simultáneamente.
Variable *relevante*	Variable utilizada en el estudio independiente y/o dependiente del dominio.

Tabla 1. Definiciones

1.5.2 Abreviaturas

Término	Significado
BDD	*Binary Decision Diagram.*
I+D+I	Investigación, Desarrollo e Innovación.
ICAPS	*International Conference on Automated Planning and Scheduling.*
IPC	*International Planning Competition*.
PDB	*Pattern DataBase*.
PDDL	*Planning Domain Definition Language*.
TFG	Trabajo Fin de Grado.
UC3M	Universidad Carlos III de Madrid.

Tabla 2. Abreviaturas

SECCIÓN 2. ESTADO DEL ARTE

Para abordar este proyecto es de vital importancia realizar inicialmente un estudio del estado del arte. Conocer la base teórica, al igual que los trabajos relacionados, para no comenzar desde cero en la medida de lo posible. La información aquí expuesta es la que resulta relevante y necesaria para la realización del proyecto, es decir, no se comentará nada que no se aplique o sea relevante para la investigación. Por otra parte, no se comentarán tampoco aspectos sobre el funcionamiento de los lenguajes de programación ni sobre los sistemas operativos utilizados, ya que son una herramienta presente en prácticamente todos los proyectos de informática, y no juegan un papel importante más allá de lo común.

2.1 Planificación automática

Dado que este TFG habla acerca de la planificación automática, es necesario entender ciertas características básicas previamente.

La planificación (Liaison, 2018) (Russell & Norvig, 2003) (UC3M, ocw, 2015) es una disciplina dentro de la inteligencia artificial que se basa en la construcción de planes (conjunto de acciones) que, partiendo desde un estado inicial, desembocan en un estado que cumple cada unas de las condiciones meta del problema a resolver. Generalmente, estos hacen uso de la búsqueda heurística para resolver los problemas. Para entender la definición, es necesario incluir algunas acepciones:

- **Estado**: es una descripción instantánea del problema. Todos los predicados e instancias presentes en un estado son ciertas, y las que no están presentes, falsas.

- **Acción**: transformación de un estado en otro. Pueden ser deterministas o no deterministas.

- **Función de coste**: para establecer el coste de realizar una acción.

- **Estado inicial**: situación de partida.

- **Conjunto de meta**: conjunto de condiciones que deben cumplirse para concluir el proceso de planificación. Pueden corresponder con uno o más estados.

- **Plan**: secuencia de acciones que, partiendo desde un estado inicial, desembocan en un *estado meta*, es decir, un estado que cumpla con todas las condiciones finales establecidas en el problema a resolver.

- **Heurística**: conocimiento que permite guiar la búsqueda.

A su vez, también es necesario definir dos elementos más que incluyen cada una de las anteriores acepciones:

- **Dominio**: descripción abstracta de una situación/mundo. Mayoritariamente, se componen de un conjunto de tipos, predicados que relacionan dichos tipos, funciones y acciones.

- **Problema**: descripción de un conjunto de objetos, un estado inicial, un conjunto de metas y una métrica (minimizar la función de coste final).

Los planificadores pueden ser, a su vez, de dos tipos:

- **Dependientes del dominio**: planificador específico para un dominio concreto.

- **Independiente del dominio**: planificador que funciona en todos los dominios de planificación.

Para llevar a cabo un proceso de planificación, es necesario utilizar un **lenguaje de planificación**, como PDDL, para modelar el dominio y el problema a resolver. A su vez, el planificador debe saber interpretar dicho lenguaje. Este planificador resuelve el problema mediante un proceso de búsqueda. La diferencia entre planificadores radica, fundamentalmente, en el proceso de búsqueda y en la definición de la heurística.

Esto implica que un plan generado pueda ser óptimo o satisfacible, dependiendo de dicha heurística y del algoritmo de búsqueda. *A** (Hart, Nilsson, & Raphael, 1968), por ejemplo, genera planes óptimos, siempre y cuando la heurística sea admisible (es decir, cuando no sobrevalora el coste real desde cualquier estado hacia la meta). Un plan óptimo es el que proporciona una serie de acciones que transitan desde el estado inicial hacia un *estado meta*, con el mínimo coste total. Dicho coste puede ser el número de acciones ejecutadas en total o el resultado de la suma de los costes obtenidos al realizar determinadas acciones en concreto. *A** será el algoritmo de búsqueda que se va a utilizar en el estudio, junto con las PDBs, un tipo de heurística admisible. Un plan satisfacible, sin embargo, se preocupa por resolver el problema, pero no asegurando devolver la solución óptima.

Para poder realizar el proceso de búsqueda, es de vital importancia interpretar la información del dominio y la del problema. Una de las formas es convertir esta información en un archivo *output.sas*, tal y como hace *Fast Downward* (Helmert, 2018). Posteriormente se explicará con más detalle. A modo de resumen, ese archivo contiene, entre otras cosas, un conjunto de variables *n*. Cada variable está asociada a un conjunto de *k* predicados y/o hechos negados o afirmados que no pueden darse al mismo tiempo. Estas variables pueden tomar un valor entre 0 y *k-1*. Cualquier estado puede traducirse a un conjunto de *n* valores numéricos. Los estados del proceso de búsqueda del planificador se componen de todos los valores de estas variables. Para que sea posible,

las acciones del dominio también se traducen para poder operar con esta nueva representación de los estados.

2.2 Problemas de búsqueda

Resulta prácticamente imposible hablar de planificación automática sin mencionar los problemas de búsqueda. Estos guardan mucha relación con las tareas de planificación, pues, de forma general, se tienen estados (incluido el inicial y las metas), acciones para transitar entre estados, una heurística, costes y la solución al problema de búsqueda. Sin embargo, esta solución puede ser tanto el conjunto de acciones necesarias, como el coste de estas o de la meta en sí.

De igual forma, la heurística empleada puede ser dependiente cómo independiente del dominio, y el algoritmo de búsqueda junto con la heurística podría dar soluciones óptimas o satisfacibles.

Dependiendo de la presencia de una heurística y de la función de coste, pueden definirse tres tipos generales de algoritmos de búsqueda:

- **Algoritmo de búsqueda no heurística sin costes**: algoritmos que no emplean ningún valor heurístico. El único coste para tener en cuenta es el número de acciones realizadas, el cual es unitario. Son algoritmos de fuerza bruta. Dos ejemplos de algoritmos de este tipo son la búsqueda en amplitud y en profundidad, ambos utilizados para obtener el camino más corto.

- **Algoritmo de búsqueda no heurística sin costes**: algoritmos que no emplean ningún valor heurístico. El coste de cada acción se traduce en el coste de los arcos entre nodos, en su representación en grafo. Un ejemplo de algoritmo de este tipo es el algoritmo de *Dijkstra*, que escoge qué acción tener en cuenta en función del coste acumulado entre los arcos por donde ha pasado, barajando muchas posibilidades en múltiples ramificaciones y caminos. Incrementa progresivamente el coste total de los subcaminos hasta dar con la solución, dado que el objetivo es minimizar el coste.

- **Algoritmo de búsqueda heurística**: estos algoritmos eligen qué acción tener en cuenta en función del coste acumulado entre los arcos por donde ha pasado más el valor heurístico de ese nodo. Dos ejemplos de algoritmos de este tipo son el *Hill Climbing* y el *A**. Ambos algoritmos son necesarios de explicar, dado que están presentes en el desarrollo de la investigación.

2.2.1 *Hill Climbing*

Este algoritmo de búsqueda heurística se lleva a cabo en los algoritmos de *Fast Downward iPDB* (Bonet, Patrik, Botea, Helmert, & Koening, 2007), *Canonical PDB* y *Zero-One PDB* (dependiendo de la configuración en estos dos últimos). No se plantea el uso de este algoritmo en la planificación en sí, pero sí en la obtención de las PDBs, dado que el resultado del proceso de búsqueda no es óptimo.

Partiendo de la definición de un algoritmo de búsqueda, *Hill Climbing* se diferencia del resto de algoritmos de búsqueda por la interpretación que hace sobre el valor de la función, pues solo tiene en cuenta el valor de la heurística en cada nodo. Dado un estado, *Hill Climbing* escogerá aquella acción que minimice el valor de la función, sin pensar a largo plazo. Solo expande los nodos de una rama que parte del estado inicial en profundidad por espacio de estados. Sin embargo, es posible que el algoritmo no encuentre ninguna solución, por lo que no es completo. Comúnmente, los resultados obtenidos suelen ser satisfacibles, obteniendo casualmente resultados óptimos.

Traduciendo esta misma filosofía a cualquier algoritmo, sea de búsqueda o no, se dirá que emplea un razonamiento de escalada o *Hill climbing* siempre y cuando tome la decisión, entre un conjunto finito de posibilidades, que le proporcione el mejor resultado en ese momento. Ese resultado podría ser el más alto, el más bajo o el que más se aproxime a cierto valor, dependiendo del problema en sí. Suele emplearse cuando lo que se busca en un resultado satisfacible y no se desea consumir excesivo tiempo en dicho proceso.

2.2.2 *A**

Este algoritmo de búsqueda heurística (Hart, Nilsson, & Raphael, 1968) se utiliza por el planificador una vez obtenido el conjunto de PDBs, para obtener un plan óptimo. Si la heurística empleada es admisible, es decir, no sobrevalora el valor del coste real desde cualquier estado hacia la meta, el resultado generado será óptimo.

Al igual que *Hill Climbing*, *A** tendrá en cuenta la acción que minimice la función total. Sin embargo, en este algoritmo se calcula sumando el coste acumulado en esa rama con el valor heurístico de ese nodo.

$$f(n) = g(n) + h(n)$$

$f(n)$ ➔ valor de la función en el nodo *n*.

$g(n)$ ➔ valor de la función de coste acumulado hasta llegar al nodo *n* desde el estado inicial.

$h(n)$ ➔ valor de la función heurística en el nodo *n*.

*A** baraja todas las alternativas desde el mínimo coste, hasta alcanzar un estado meta. En resumidas cuentas, el valor de la función le indica qué nodo expandir, y este será el que lo minimice. En la siguiente iteración, vuelve a tener en cuenta todos los nodos alcanzables desde los estados actuales, y vuelve a expandir el más prometedor. Así progresivamente. Cuando finalmente expande la meta, el camino de nodos expandidos desde la meta hasta el estado inicial será la solución óptima. En caso de querer obtener más de una solución óptima, bastaría con no detener el algoritmo al expandir la meta, sino dejarlo ejecutar hasta que el coste acumulado de un conjunto de nodos expandidos supere el coste óptimo. La diferencia fundamental con *Hill Climbing* es que este último solo considera los sucesores del nodo recién expandido, mientras que *A** considera todos los sucesores generados y no expandidos.

En caso de que la heurística siempre proporcione valor 0, *A** actuará igual que el algoritmo *Dijkstra*. Si además el coste de cada acción siempre es unitario, *A** actuará igual que un algoritmo de fuerza bruta en amplitud.

2.3 PDDL

Los dominios y los problemas están modelados en PDDL, por lo que es necesario entender su sintaxis y su funcionamiento.

PDDL son las siglas de *Planning Domain Definition Language* (McDermott, 1998). Es decir, es el lenguaje común para definir dominios de planificación, aunque también es capaz de definir problemas. Este lenguaje es el que soporta y maneja la mayor parte de planificadores, incluyendo el que proporciona *Fast Downward*. A continuación, se explicará los rasgos básicos y necesarios de su nomenclatura.

2.3.1 Dominio

Estructura básica del dominio:

(define (domain <nombre del dominio>)

(:requirements <lista de requirements>)

(:types <lista de tipos>)

(:constants <lista de constantes>)

(:predicates <lista de predicados>)

(:functions <funciones>)

(:action <descripción de la acción número 1>) (:action <descripción de la acción número 2>) … (:action <descripción de la acción número n>))

Los elementos anteriores en negrita podrían no aparecer, dependiendo de la descripción del dominio. Además, los comentarios se podrán redactar escribiendo "*;*" previamente al comentario en cuestión. A continuación, se desglosará y se comentará cada parte de la estructura del dominio:

- ***Requirements***. Alberga las características del dominio, aunque no es del todo necesario incorporarlas. El código podría ser correcto e interpretarse bien sin necesidad de añadirlos, dependiendo del planificador en cuestión, aunque es aconsejable incorporarlas siempre. Las más básicas son:

  - ⇨ ***:strips*** - Define que se pueden añadir y borrar predicados en los efectos de las acciones, e incorporar precondiciones en estas.
  - ⇨ ***:equality*** - Indica que el dominio usa el símbolo "=" interpretado como igualdad.
  - ⇨ ***:typing*** - Indica que se pueden añadir tipos: ***(:types <lista de tipos>)***.
  - ⇨ ***:conditional-effects*** - Indica que se puede añadir el símbolo *when* en los efectos de una acción para poder definir diversos comportamientos en una misma acción.
  - ⇨ ***:action-costs*** / ***:fluents*** - Indica que se pueden añadir funciones: ***(:functions <funciones>)***. Se usará uno u otro dependiendo del planificador. En *Fast Downward* se usa *action-costs*.
  - ⇨ ***:negative-preconditions*** - Indica que el dominio usa el símbolo *not* interpretado como negación en las precondiciones de una acción.

- ***Types***. Se define la jerarquía de tipos. Ejemplo:

  (:types

  comida – object

  fruta verdura – comida)

  object es la raíz de la jerarquía de tipos. Se podría prescindir de él.

- ***Constants***. Declaración de objetos constantes que van a aparecer en cualquier problema. Suelen añadirse en el dominio cuando hay una acción que requiera que un objeto sea igual a ese valor constante. El resto de declaración de objetos se hace dentro del problema en cuestión. En caso de declarar una constante de un tipo en cuestión, este se introduce tras un guión. Ejemplos: *(:constants cero – numero)*, *(:constants cero)*.

- ***Predicates***. Es el conjunto de estructuras que sirven para definir un estado. Todas las instancias de los predicados presentes en un estado representan la realidad en dicho estado, y aquellas instancias de los predicados que no aparecen son falsas. Los predicados se pueden declarar de distintas formas:

- ⇨ **Predicado sin objetos**: solo puede instanciarse una vez en un problema, y puede ser verdad o mentira. Ejemplo: *(esta_leyendo_receta)*.
- ⇨ **Predicado con objetos sin tipo**: puede instanciarse, como máximo, tantas veces como objetos y constantes se han definido en el problema, siempre y cuando no se definan tipos en el dominio. Ejemplo: *(pelar ?alimento)*; con 3 objetos en el problema, que son *naranja*, *pera* y *puerro*; los predicados posibles instanciados podrían ser *(pelar naranja)*, *(pelar pera)* y *(pelar puerro)*. De igual forma, podrían crearse predicados que hagan uso de más de un objeto.
- ⇨ **Predicado con objetos con tipo**: puede instanciarse, como máximo, tantas veces como objetos y constantes de ese mismo tipo se han definido en el problema. Ejemplo: *(pelar ?alimento – fruta)*; con 3 objetos ciudades en el problema que son *naranja*, *pera* y *puerro*, los cuáles solo dos de ellos son frutas; los predicados posibles instanciados podrían ser *(pelar naranja)* y *(pelar pera)*, pero no *(pelar puerro)*. De igual forma, podrían crearse predicados que hagan uso de más de un objeto.

- ***Functions***. Son relaciones entre objetos, donde uno de ellos es de tipo numérico. En caso de no existir ninguna función, se supondrá que el coste por acción es de uno, y el objetivo es de minimizar el coste total del plan, es decir, reducir el número de acciones. La sintaxis básica es: *(:functions (total-cost) – number)*. Sin embargo, pueden definirse las funciones siguiendo la misma estructura que los predicados.

- ***Action***. Las acciones tienen la siguiente estructura básica: *(:action <nombre_accion> :parameters (<lista_parámetros>) :preconditions (and <lista_precondiciones>) :effects (and <lista_efectos>))*

- ⇨ **Parámetros**: Los parámetros se introducen declarando variables y el tipo de dichas variables (si se desease especificar el tipo). Ejemplos: *(?alimento1 – fruta ?alimento2 – verdura)*. La acción con dichos parámetros usará dos objetos ciudad, que se nombrarán en las precondiciones y/o en los efectos.
- ⇨ **Precondiciones**: son una lista de predicados, predicados negados y condiciones. Estas se pueden expresar de distintas formas:
  - **Predicado sin objeto**: por ejemplo *(esta_leyendo_receta)*.
  - **Predicado negado sin objeto**: por ejemplo *(not (esta_leyendo_receta)).*
  - **Predicado con objeto**: por ejemplo *(pelar ?alimento1)*, donde *alimento1* es una variable definida en los parámetros.
  - **Predicado negado con objeto**: por ejemplo *(not (pelar ?alimento1*)), donde *alimento1* es una variable definida en los parámetros.
  - **Comparación entre objetos**: son condiciones que deben cumplir los objetos instanciados en los parámetros, por ejemplo: *(not (= ?objeto1 ?objeto2))*, es decir, que el *objeto1* no debe ser igual a *objeto2*.
  - **Comparación entre funciones**: son condiciones que deben cumplir las funciones, por ejemplo: *(>= (funcion1 ?objeto1) (funcion2 ?objeto2))*, es

decir, que el resultado numérico de la función denominada *funcion1* debe ser mayor o igual que el resultado numérico de la función denominada *funcion2*.

Para poder ejecutar los efectos de la acción, deben poder instanciarse objetos en todas las precondiciones y cumplirlas. Es decir, que al menos exista una combinación de objetos que, al introducirla en las variables de los parámetros, muestren predicados que están presentes en el estado y predicados negados que no existan en el estado en cuestión, cumpliendo con los criterios de comparación entre objetos y funciones. El conjunto de todas las acciones que se pueden ejecutar combinadas con los objetos que se pueden introducir en los parámetros conforman el conjunto conflicto. La heurística implementada ayuda a solventarlo y escoger una acción instanciada en concreto.

- ⇨ **Efectos**: en caso de que se puedan instanciar objetos en las precondiciones y cumplir todas las premisas, se llevarán a cabo los efectos. En los efectos se pueden añadir y/o eliminar predicados y modificar funciones:
  - **Predicado con o sin objeto sin negación *not***: este se añadirá al estado, creando otro nuevo estado.
  - **Predicado con o sin objeto acompañado de la negación *not***: este se eliminará del estado, creando otro nuevo estado.
  - **Modificar funciones**: pueden ser de incremento o decremento. La estructura básica es: *(increase/decrease (<nombreFuncion>) <cantidad a incrementar o decrementar>)*. Esa cantidad puede reflejarse en un número o en el valor numérico de otra función. Por ejemplo: *(increase (total-cost) 1)*; o *(decrease (total-cost) (valor_incrementar))*.
  - **Estructuras *when***: se comportan como una acción dentro de los efectos de una acción del dominio. Su estructura básica es: *(when <precondiciones> <efectos>)*. En caso de que se vaya a ejecutar una acción, si esta cumple también la precondición del *when*, llevará a cabo más cambios en la base de hechos. Sin embargo, en caso de no cumplir dichas precondiciones, se ejecutarán de igual forma los efectos, pero no los efectos del *when*. Ejemplo: *(when (and (precondicion1 ?p1) (precondicion2 ?p2)) (and (efecto1 ?e1) (not(efecto2 ?e2)))*.

2.3.2 Problema

Estructura básica del problema:

(define (problem <nombre del problema>)

(:domain <nombre del dominio asociado>)

(:objects <lista de objetos>)

(:init <lista de instancias de predicados iniciales>)

(:goal <lista de presencia y/o ausencia de instancias de predicados para definir la condición meta>)

(:metric minimize/maximize (<nombre_funcion_objetivo>)) *)*

Los elementos anteriores en negrita podrían no aparecer, dependiendo de la descripción del dominio. Además, los comentarios se podrán redactar escribiendo *";"* previamente al comentario en cuestión. A continuación, se desglosará y se comentará cada parte de la estructura del dominio:

- ***Objects***. Los objetos se definen declarando los nombres de cada uno de ellos, determinando al final de la línea el tipo de estos. En caso de no establecer el tipo, bastaría con declarar su nombre:

  - ⇨ Ejemplo de declaración de objetos teniendo en cuenta el tipo: *(:objects naranja pera - fruta)*.
  - ⇨ Ejemplo de declaración de objetos sin tener en cuenta el tipo: *(:objects pera naranja puerro)*.

- ***Init***. Se declara el estado inicial, es decir, todos los predicados iniciales instanciados. Por esa razón, todos están afirmados, ya que la no inclusión de un predicado instanciado supone la inclusión negada de este *(not)*. Por ejemplo: *(:init (querer_cocinar_ensalada) (sin_lavar lechuga) (sin_lavar tomate))*. También, en caso de incluir funciones en el dominio, puede establecerse aquí su valor inicial. Ejemplo: *(= (total-cost) 0)*.

- ***Goal***. Condiciones que deben cumplirse para establecer un estado meta. Por tanto, el conjunto de metas es mayor o igual que uno, dado que podría darse el caso de que exista más de un posible estado que cumpla todas las condiciones meta. Además, en este caso, podrían incluirse instancias de predicados negados, ya que la no presencia de alguno de estos no supone que deban ser falsos, sino que no es relevante su presencia para establecer que un estado es meta o no. Ejemplo: *(:goal (and (ensalada_cocinada) (not(esta_leyendo_receta))))*.

- ***Metric***. Métrica del problema. Suele ser minimizar. Podría no incluirse si no se han declarado funciones en el dominio asociado. Ejemplo: *(:metric minimize (total-cost)).*

2.4 PDBs

El tipo de heurística que se va a estudiar se llama PDB (Culberson & Schaeffer, 1998), siglas de *Pattern DataBase*, y al contrario que PDDL y planificación automática, este no

se ha impartido en mis cuatro años de carrera. Fue necesario comprenderlo antes de empezar con la investigación. Este tipo de heurística lo maneja *Fast Downward*.

Estas nacen de la problemática de querer elaborar una heurística perfecta teniendo en cuenta todas las variables. La explosión combinatoria implica un enorme gasto de memoria y de tiempo al generar todas las transiciones mediante todas las combinaciones de variables.

Una PDB es una simplificación de todo el conjunto de variables. Dado que es una simplificación, el gasto en tiempo y memoria en generarlas disminuye considerablemente. A partir de estas variables debidamente escogidas, se elaborará un grafo, otorgándole a cada estado un valor heurístico. Como resulta evidente, un estado del grafo corresponderá a más estados en el problema original, tantos como combinatorias se puedan formar entre todas las variables que no se han elegido para formar dicha PDB. El problema que subyace detrás de esta idea reside en la elección de variables a la hora de formar una PDB. Depende de dicha elección, la PDB resultante podría ser más o menos informada.

Con el objetivo de refinar la heurística, se podría crear una colección de PDBs. La heurística resultante pasaría a ser el máximo de todas las PDBs en cada estado. Como es bien sabido, cuando *n* heurísticas son admisibles, el máximo de todas ellas también lo es. Este conjunto de PDBs estará conformado por patrones que tengan un valor máximo en, al menos, un estado. Si se diese el caso de que una PDB tuviese valores tan bajos como para no salir nunca candidata al realizar el máximo, esa PDB terminaría podándose, es decir, eliminándose, pues no aporta nada por sí misma.

Por otra parte, existen PDBs que podrían funcionar mejor junto a otras PDBs, pues sus heurísticas se podrían sumar. A estas PDBs se le denominan PDBs aditivas. Esto podrá suceder siempre y cuando no exista ninguna acción que, tras ejecutarla, suponga un tránsito en el grafo de una sola PDB. Por tanto, cuando una PDB marque un valor numérico *k* para un determinado estado del grafo y otra PDB (aditiva con la primera) marque un valor *l*, dado que no existe una acción que haga transitar en las dos PDBs a la vez, al menos se podrá llegar a la meta con *k+l* movimientos.

Por último, como existen PDBs aditivas, una misma selección de PDBs podría dar origen a más de un conjunto de patrones, dependiendo de cómo se asocien las PDBs aditivas entre sí. Por ejemplo, podría existir una PDB que sea aditiva con otra segunda y con otra tercera, pero no de forma simultánea. Además, en este ejemplo, en algunos estados se podrían obtener valores heurísticos más informados mediante la suma de las dos primeras PDBs y, en otros estados, calcular valores heurísticos más elevados mediante la otra suma de las otras dos heurísticas (es decir, la primera PDB con la tercera PDB aditiva).

Dicho esto, el valor heurístico de un conjunto de PDBs en un determinado estado es igual al valor máximo de todas las heurísticas en dicho estado (una por cada PDB) y de todas las sumas obtenidas por las PDBs aditivas en ese mismo estado.

$$h(estado) = max(\, h(PDB1), h(PDB2), \dots, h(PDBn), \sum_{i = PDBs\ del\ conjunto\ aditivo\ 1} h(i), \sum_{i = PDBs\ del\ conjunto\ aditivo\ 2} h(i), \dots, \sum_{i = PDBs\ del\ conjunto\ aditivo\ m} h(i)\ )$$

2.5 Fast Downward

Fast Downward (Helmert, 2018) Es una plataforma que permite la construcción de múltiples planificadores. Con esta plataforma se va a realizar la investigación. Es necesario saber cómo utilizarlo, comprender en qué se basa y entender cómo implementa ciertos algoritmos de cálculo de heurísticas.

Fast Downward, a su vez, proporciona múltiples algoritmos. Está implementado en *Python* y en C++. Puede utilizarse desde *Ubuntu*, *Mac OS X* o *Windows*. Sin embargo, se recomienda su uso en *Ubuntu*. El código fuente que proporciona se puede modificar y recompilar, lo que favorece los proyectos de investigación.

Los algoritmos de cálculo de heurísticas que se van a utilizar trabajan con PDBs. Estos son: *Canonical PDB*, *iPDB* (Bonet, Patrik, Botea, Helmert, & Koening, 2007) y *Zero-One PDB*. Por otra parte, el algoritmo *PDB* tan solo maneja patrones individuales, por lo que no es demasiado conveniente utilizarlo. A continuación, se explica brevemente la estructura de funcionamiento general de estos 3 algoritmos:

En primer lugar, el traductor de *Fast Downward* construye el archivo *output.sas*, que es el que contiene los datos del dominio y del problema juntos, para poder interpretarlo. En segundo lugar, construye las PDBs usando el método concreto en cuestión (*combo*, *Hill climbing*, etc.), todos ellos accesibles desde alguno de los tres algoritmos anteriormente citados. Finalmente, interpretando las PDBs obtenidas, se ejecuta el planificador, llegando o no a una solución, es decir, un plan.

2.5.1 *output.sas*

Este archivo es una traducción del dominio y del problema, pasando a tener variables y relaciones entre estas. Cada variable está asociada a un conjunto de predicados y/o hechos negados o afirmados que no pueden darse al mismo tiempo. Estas variables

pueden tomar un valor numérico, asociándolo a uno de los predicados instanciados que lo contienen. Por tanto, cualquier estado del proceso de búsqueda del planificador puede traducirse a un conjunto de valores numéricos, tantos como variables haya. Para que sea posible, las acciones del dominio también se traducen para poder operar con esta nueva representación de los estados. A continuación, se comentará el contenido del fichero punto por punto:

- **Versión y métrica**. En primer lugar, se indica la versión del traductor, y la métrica, que es 0 cuando no se usan costes y 1 en caso contrario.

- **Número de variables**.

- **Cada una de las variables**. Estas se dividen en 4 partes fundamentales:

  - ⇨ Número de la variable.
  - ⇨ La etiqueta de axioma: Indica el orden de evaluación de las reglas de los axiomas. Ese valor no se va a interpretar en el actual proyecto, pues mayoritariamente es -1.
  - ⇨ Número de predicados instanciados en la variable.
  - ⇨ Predicados instanciados: todos los predicados no pueden darse a la vez. Cuando una variable toma un valor numérico, se refiere al predicado instanciado que ocupa esa posición. Por ejemplo, si la variable 0 tuviese 4 predicados (*Atom estoy_en (posicion1), Atom estoy_en (posicion2), Atom estoy_en (posicion3), Atom estoy_en (posicion4)*), y en un estado tuviese el valor 1, este señalaría al predicado *estoy_en (posicion2)*. En caso de que se use *NegatedAtom* en lugar de *Atom*, querrá decir que no aparece dicho predicado instanciado.

- **Número de *mutex***.

- **Cada uno de los *mutex***. Indica qué predicados instanciados no pueden darse simultáneamente. No se han introducido dentro de la misma variable ya que pertenecen a variables distintas, y al agrupar todos los predicados en la misma se daría el caso de que dos predicados instanciados sí pudiesen suceder a la vez. Se componen de 2 partes:

  - ⇨ Número de predicados instanciados asociados en los *mutex*.
  - ⇨ Cada uno de los predicados instanciados del *mutex*: estos tienen dos valores numéricos. El primero hace referencia a la variable, y el segundo al valor de dicha variable, es decir, al predicado instanciado en cuestión.

- **Estado inicial**. Se compone de tantos números como variables haya. Cada número hace referencia a un predicado instanciado de cada una de las variables, traduciendo el estado inicial en PDDL en un conjunto de dígitos.

- **Metas**. Dado que las metas pueden agrupar varios posibles estados, no se puede definir como el estado inicial, es decir, mediante un conjunto de *n* valores, siendo *n* el número de variables. Por ello, la estructura que sigue es semejante a la de los *mutex*, que es la siguiente:

  - ⇨ Número de variables meta.
  - ⇨ Cada uno de los predicados instanciados que hacen referencia a la meta: estos tienen dos valores numéricos. El primero hace referencia a la variable, y el segundo al valor de dicha variable, es decir, al predicado instanciado en cuestión.

- **Número de operadores**.

- **Cada uno de los operadores**. Es el resultado de instanciar todas las acciones junto con todos los objetos posibles. Se componen de 6 partes. No será necesario entender su estructura dado que no se va a utilizar en el proyecto. A pesar de ello se resume a continuación:

  - ⇨ Acción y objetos en PDDL.
  - ⇨ Número de precondiciones.
  - ⇨ Cada una de las precondiciones: con la misma nomenclatura que las metas, es decir, el número de la variable junto al valor de dicha variable.
  - ⇨ Número de efectos.
  - ⇨ Cada uno de los efectos: estos se componen a su vez de varios valores, que se enumeran a continuación:
    - Cantidad de condiciones de efecto. En dominios *STRIPS*, generalmente es 0.
    - Un par de números variable/valor.
    - Número que indica la variable afectada por el efecto.
    - Valor que debe tener esa variable. Será -1 si no hay ningún valor en concreto que la variable deba tener.
    - Nuevo valor para la variable afectada.
  - ⇨ El coste del operador: es equivalente al coste de la acción asociada.

2.5.2 *iPDB*

iPDB (Bonet, Patrik, Botea, Helmert, & Koening, 2007) es un algoritmo de obtención de PDBs. A la hora de consultar el valor de la heurística para un estado, lo interpreta de acuerdo con lo comentado al final del apartado 2.4 PDBs: “el valor heurístico de un conjunto de PDBs en un determinado estado es igual al valor máximo de todas las heurísticas en dicho estado (una por cada PDB) y de todas las sumas obtenidas por las PDBs aditivas en ese mismo estado”. Esto se realiza con todos los conjuntos de PDBs.

Este algoritmo obtiene las PDBs de forma iterativa mediante *Hill Climbing*. A continuación, se explica punto por punto su funcionamiento a modo de resumen:

1. En primer lugar, identifica las variables con instanciaciones de predicados meta. Una vez hecho esto, crea una PDB con cada una de ellas y las incluye en el conjunto.
2. A partir de este punto, comienza el proceso iterativo.
    a. Por cada PDB en el conjunto, se crea una PDB auxiliar producto de coger dicho patrón y añadirle una variable que refine el grafo. Es decir, no se incorporará ninguna variable que no mejore el valor heurístico del grafo en sí, pues se considerará que esa combinación de variables no aporta nada.
    b. Cuando la PDB auxiliar se crea, se construye el grafo y se evalúa con un conjunto previamente definido de ejemplos. Por cada ejemplo en el que el grafo proporciona valores más altos de heurística al introducir esta nueva variable, se incrementará en una unidad un contador. También es posible que la PDB sea tan larga que no quepa en memoria. En casos como esos, se descarta automáticamente.
    c. Si el valor del contador supera al contador de la PDB auxiliar en esa iteración con el valor más alto, se seleccionará como futura PDB la actual PDB auxiliar.
    d. Cuando la iteración termine, si el contador más alto supera un valor mínimo, se añadirá la PDB auxiliar al conjunto de PDBs. Como se considerarán aquellos valores más altos y se incluirán directamente dichas PDBs en la colección, el proceso de búsqueda que subyace detrás de este algoritmo se dice que es *Hill Cimbing* o de escalada.
3. Cuando todas las PDBs se hayan construido, si se diese el caso de que una PDB tuviese valores tan bajos como para no salir nunca candidata al realizar el máximo, esa PDB terminaría podándose, es decir, eliminándose, pues no aporta nada per se.
4. Por último, antes de lanzar el planificador con esta heurística e interpretarla según lo comentado anteriormente, se identifican las PDBs aditivas entre sí, y se forman tantos conjuntos como relaciones se pueden crear entre estas PDBs.

Al llamar al algoritmo, se pueden introducir ciertos parámetros. Entre los relevantes se destacan los siguientes:

- ***pdb_max_size***: Es el máximo número de estados por PDB. Por defecto vale 2000000.

- ***collection_max_size***: Es el máximo número de estados por colección. Por defecto vale 20000000.

- ***num_samples***: Es el número de ejemplos empleados para evaluar una PDB auxiliar. Por defecto vale 1000.

- ***min_improvement***: Es el mínimo valor que debe dar el máximo valor del contador en una PDB para decidir si introducirla en la colección o detener el proceso de búsqueda de PDBs. Por defecto vale 10.

- ***max_time***: Tiempo máximo (en segundos) que debe durar el algoritmo de generación de PDBs. Por defecto vale infinito.

- ***random_seed***: es la semilla para generar números aleatorios. Por defecto vale -1, para aludir al generador de números aleatorios globales.

- ***max_time_dominance_pruning***: el tiempo máximo (en segundos) dedicado a la poda de PDBs, a la generación de PDBs aditivas y, por ende, a la generación de conjuntos de conjuntos de patrones. Por defecto vale infinito.

- ***transform***: correspondiente a transformaciones opcionales de tareas para la heurística. Por defecto es *no_transform()*.

- ***cache_estimates***: Estimaciones de heurísticas de *caché*. Por defecto vale *true*.

2.5.3 *Canonical PDB*

Hace referencia a la interpretación de la heurística realizada por iPDB: "el valor heurístico de un conjunto de PDBs en un determinado estado es igual al valor máximo de todas las heurísticas en dicho estado (una por cada PDB) y de todas las sumas obtenidas por las PDBs aditivas en ese mismo estado". Esto se realiza con todos los conjuntos de PDBs.

Al llamar al algoritmo, se pueden introducir ciertos parámetros. Entre los relevantes se destacan los siguientes:

- ***patterns***: Es la forma de obtener el conjunto de PDBs. Por defecto es *systematic(1).* A continuación, se describe cada una de las posibilidades:

  - *Combo*: Este algoritmo de colección de patrones pretende hacer uso de la mayor cantidad de variables en un solo patrón. Para ello, identifica las variables relacionadas con las metas, y las va introduciendo en una PDB. Dado que dichas variables se suelen encontrar en las últimas posiciones, el orden de inserción es desde la última variable hasta la primera. En caso de terminar, incluiría más variables no meta siguiendo este orden hasta que la memoria llegue a su límite e impida meter la siguiente variable. En caso de que no se puedan incluir todas las variables meta en una PDB, se crean PDBs individuales con una variable meta en cada una, hasta finalizar. Además, se puede introducir un parámetro:
    - *Max_states*: tamaño máximo de abstracción. Por defecto vale 1000000.

- ⇨ Patrones generados por algoritmos genéticos: como su propio nombre indica, se trate de generar PDBs mediante algoritmos genéticos. Debido a que el código fuente no funciona debidamente en ciertos dominios, no se abordará en el proyecto. Suele mandar un mensaje de error cuando los patrones obtenidos contienen muchas variables, y no llega a mostrar ninguna PDB. Se suele quedar estancado sin dar ninguna solución. También, en ocasiones, muestra un mensaje de error a la hora de podar. Sería muy poco sistemático ejecutarlo solo en ciertos dominios. En trabajos futuros podría considerarse, siempre y cuando se asegure su funcionamiento en todos los dominios con los que se va a trabajar.
- ⇨ *Hill Climbing*: este algoritmo es el mismo que el de generación de patrones de *iPDB*. Además, sus parámetros son los mismos, exceptuando *max_time_dominance_pruning*, *transform* y *cache_estimates*, que ya los incluye canonical PDB.
- ⇨ Patrones manuales: lista de patrones introducidos manualmente en forma de lista.
- ⇨ Generación de patrones sistemática: este algoritmo de creación de patrones forma toda la combinatoria entre variables, construyendo un patrón por cada combinación de variables. Se pueden introducir 2 parámetros:
    - *pattern_max_size*: muestra el número de variables por patrón como máximo. Por defecto es 1. Por ejemplo, si el número de variables por patrón máximo es de 3 y se tienen *n* variables en total, el número de patrones que se forman (cuando *only_interesting_patterns* es *false*), es $n + n*(n-1) + n*(n-1)*(n-2)$.
    - *only_interesting_patterns*: indica si solo se va a realizar la unión de patrones que aportan información juntos, es decir, realizar la unión si esta consigue aportar más información que los patrones individuales. Por ejemplo, no se incorporan patrones de tamaño uno con una variable meta, pues no aporta información a la heurística. Por tanto, si este parámetro toma el valor true (por defecto), la combinación que muestra equivale a los patrones auxiliares que *iPDB* tiene en cuenta a la hora de elegir los patrones en su algoritmo de creación de PDBs.

- ***max_time_dominance_pruning***: el tiempo máximo (en segundos) dedicado a la poda de PDBs, a la generación de PDBs aditivas y, por ende, a la generación de conjuntos de conjuntos de patrones. Por defecto vale infinito.

- ***transform***: correspondiente a transformaciones opcionales de tareas para la heurística. Por defecto es *no_transform().*

- ***cache_estimates***: Estimaciones de heurísticas de *caché*. Por defecto vale *true*.

2.5.4 *Zero-One PDB*

Este algoritmo, al contrario que *iPDB*, no calcula su valor heurístico mediante el valor máximo de todas sus PDBs y aquellas aditivas. El cálculo de la heurística se lleva a cabo mediante la forma más básica de partición de costes. Esta consiste en sumar simplemente los valores heurísticos de todos los patrones de la colección, empezando por el primero. Sin embargo, si un operador afecta a más de un patrón, solo se tiene en cuenta el coste del primero de todos ellos, para garantizar la aditividad de los patrones. Es por eso por lo que el orden de los patrones es importante en este algoritmo porque, entre otras cosas, no realiza poda.

Al llamar al algoritmo, se pueden introducir ciertos parámetros. Entre los relevantes se destacan los siguientes:

- ***patterns***: Es la forma de obtener el conjunto de PDBs. Por defecto es *systematic(1).* A continuación, se describe cada una de las posibilidades:

  - ⇨ *Combo*: igual que en *Canonical PDB*.
  - ⇨ Patrones generados por algoritmos genéticos: igual que en *Canonical PDB*.
  - ⇨ *Hill Climbing*: igual que en *Canonical PDB*. Sin embargo, la razón por la que el parámetro *max_time_dominance_pruning* no aparece no es porque lo incluya *Zero-One*, si no porque este algoritmo no realiza poda de PDBs.
  - ⇨ Patrones manuales: igual que en *Canonical PDB*.
  - ⇨ Generación de patrones sistemática: igual que en *Canonical PDB*.

- ***transform***: correspondiente a transformaciones opcionales de tareas para la heurística. Por defecto es no*_transform()*.

- ***cache_estimates***: Estimaciones de heurísticas de *caché*. Por defecto vale *true*.

2.6 IPC

IPC (Keller, Sanner, & Say, 2018) recoge muchos dominios que podrían resultar interesantes de analizar, y es aquí donde se han extraído los 5 dominios que se van a analizar y todos sus problemas de optimalidad.

IPC son las siglas de *International Planning Competition*. Esta competición está organizada por ICAPS (Liaison, 2018), *International Conference on Automated Planning and Scheduling*. Evalúa de forma empírica los sistemas de planificación mediante una serie de problemas establecidos. De esta forma, promueven la investigación de la planificación, destacan desafíos frente a la comunidad y proporcionan nuevos problemas útiles para futuras investigaciones.

Hasta el momento, se han organizado 9 competiciones internacionales de planificación por ICAPS. Los dominios que se han utilizado para el estudio se han extraído de las dos últimas competiciones, la de 2014 y la de 2018.

2.6.1 Dominio *Snake*

Este dominio trata acerca del juego *Snake*. Pertenece a los dominios del *track clásico* (determinista) de la competición IPC del año 2018, y viene acompañado de 20 problemas de optimalidad.

Esta versión trata acerca de una serpiente que debe comerse todos los puntos de comida sin chocarse con los límites del tablero o con ella misma. Se disponen todos los elementos en dicho tablero. La parte de la serpiente que se maneja es la cabeza. El resto del cuerpo simplemente sigue el recorrido de su cabeza. Por cada punto de comida ingerido, la serpiente crece una unidad de longitud. Inicialmente, la serpiente tiene una posición y un tamaño definidos, y hay varios puntos de comida ubicados en el tablero. Por cada punto de comida ingerido, aparece otro en otra posición predefinida. Cuando todos sean devorados, finalizará.

- Implementación del dominio:

- ⇨ ***Requirements***:
  - *strips*.
  - *Negative-preconditions*.
- ⇨ **Constantes**:
  - ***Dummypoint***: punto de comida ficticio para poder verificar que la serpiente se ha comido todos los puntos de comida.
- ⇨ **Predicados**:
  - ***(ISADJACENT ?x ?y)***: adyacencia entre las dos casillas *x* e *y*. Es un predicado constante.
  - ***(tailsnake ?x)***: posición de la cola de la serpiente.
  - ***(headsnake ?x)***: posición de la cabeza de la serpiente.
  - ***(nextsnake ?x ?y)***: puntos consecutivos del cuerpo de la serpiente.
  - ***(blocked ?x)***: Posición bloqueada por la serpiente.
  - ***(ispoint ?x)***: Posición con punto de comida.
  - ***(spawn ?x)***: Posición en la cual aparecerá el siguiente punto de comida.
  - ***(NEXTSPAWN ?x ?y)***: Posición donde aparecerá el punto de comida *x* después de comer el punto de comida *y*. Es un predicado constante.
- ⇨ **Acciones**:
  - ***move***: la cabeza de la serpiente se desplaza a una casilla adyacente sin comida y no bloqueada. Cuando esto ocurre, toda la serpiente queda

intacta a excepción de la cola de la serpiente, que abandona su actual posición y pasa a ser la siguiente posición que la une.
    - ***move-and-eat-spawn***: la cabeza de la serpiente se desplaza a una casilla adyacente con comida y no bloqueada. Cuando esto ocurre, la cola de la serpiente se mantiene en su sitio, haciendo que la serpiente crezca una unidad de longitud. Además, el punto de comida desaparece y aparece un nuevo punto de comida siguiendo el orden de aparición de estas, determinado por *NEXTSPAWN* y por *spawn*.
    - ***move-and-eat-spawn***: la cabeza de la serpiente se desplaza a una casilla adyacente con comida y no bloqueada. Cuando esto ocurre, la cola de la serpiente se mantiene en su sitio, haciendo que la serpiente crezca una unidad de longitud. El punto de comida desaparece, pero no aparece un nuevo punto de comida ya que se ha alcanzado el *dummypoint*.

- Implementación de los problemas:

- ⇨ **Objetos**:
    - Posiciones del tablero de juego.
- ⇨ **Predicados iniciales**:
    - ***ISADJACENT***: con todas las posiciones del tablero de juego.
    - ***tailsnake***: solo un predicado, ya que solo hay una cola.
    - ***headsnake***: solo un predicado, ya que solo hay una cabeza.
    - ***nextsnake***: con todas las posiciones donde hay una serpiente, incluyendo la cola y la cabeza.
    - ***blocked***: con todas las posiciones donde hay una serpiente, incluyendo la cola y la cabeza.
    - ***ispoint***: uno por cada posición con un punto de comida.
    - ***spawn***: un solo predicado, con la posición en la cual aparecerá el siguiente punto de comida.
    - ***NEXTSPAWN***: con todo el orden de aparición de los puntos de comida.
- ⇨ **Predicados meta**:
    - ***Ispoint***: predicados negados con todas las posiciones donde existirá un punto de comida.

2.6.2 Dominio *Termes*

Este dominio pertenece a los dominios del *track clásico* (determinista) de la competición IPC del año 2018, y viene acompañado de 20 problemas de optimalidad.

El dominio *Termes* trata acerca de un robot de construcción. A partir de un tablero sin altura con una cantera previamente ubicada, se debe acabar con unos bloques colocados en una posición específica. Esto tiene varias restricciones. Los bloques podrá

cogerlos y dejarlos tantas veces como quiera de la cantera, que tiene provisión de bloques de manera indefinida. Podrá moverse hacia adelante, atrás, hacia la izquierda y hacia la derecha siempre y cuando esté sobre una superficie de la misma altura o una unidad superior o inferior. Solo podrá colocar un bloque en una posición adyacente a la actual siempre y cuando esté a la misma altura.

- Implementación del dominio:

- ⇨ ***Requirements*:**
  - *typing*.
  - *Negative-preconditions*.
- ⇨ **Tipos**:
  - ***numb***: número.
  - ***Position***: posición.
- ⇨ **Predicados**:
  - ***(height ?p - position ?h - numb)***: altura de una posición.
  - ***(at ?p - position)***: posición del robot.
  - ***(has-block)***: predicado que indica si el robot posee un bloque o no.
  - ***(SUCC ?n1 - numb ?n2 - numb)***: Sucesión entre dos números. Es un predicado constante.
  - ***(NEIGHBOR ?p1 - position ?p2 - position)***: adyacencia entre las dos casillas *p1* e *p2*. Es un predicado constante.
  - ***(IS-DEPOT ?p - position)***: indica la posición de la cantera. Es un predicado constante.
- ⇨ **Acciones**:
  - ***move***: el robot se mueve una casilla adyacente a la actual que tiene la misma altura.
  - ***move-up***: el robot se mueve una casilla adyacente a la actual que tiene una altura de una unidad superior a la posición actual.
  - ***move-down***: el robot se mueve una casilla adyacente a la actual que tiene una altura de una unidad inferior a la posición actual.
  - ***place-block***: si el robot se encuentra en una posición determinada y sostiene un bloque, puede dejarlo en una casilla adyacente que tenga la misma altura (y que no sea la cantera), logrando que la altura de dicha posición crezca una unidad y dejando de sostener ese bloque.
  - ***remove-block***: si el robot se encuentra en una posición determinada y no sostiene un bloque, puede reducir la altura de una casilla adyacente que esté una unidad por encima, logrando que la altura de dicha posición decrezca una unidad y pasando a sostener un bloque.
  - ***create-block***: si el robot se encuentra en la cantera y no tiene un bloque, puede coger uno.
  - ***destroy-block***: si el robot se encuentra en la cantera y tiene un bloque, puede dejarlo y no sostener ninguno.

- Implementación de los problemas:

- ⇨ **Objetos**.
  - ***numb***: números desde el 0 hasta la altura máxima que puede alcanzar una casilla.
  - ***position***: cada una de las posiciones del tablero.
- ⇨ **Predicados iniciales**:
  - ***height***: altura inicial de cada una de las posiciones del tablero.
  - ***at***: un predicado con la posición inicial del robot.
  - ***SUCC***: sucesión de todos los números del problema.
  - ***NEIGHBOR***: con todas las posiciones del tablero de juego.
  - ***IS-DEPOT***: un predicado indicando la posición de la cantera.
- ⇨ **Predicados meta**:
  - ***height***: alturas finales de cada una de las casillas del tablero.
  - ***has-block***: indicando que en las metas el robot no debe sostener ningún bloque.

2.6.3 Dominio *Hiking*

El dominio *Hiking* pertenece a los dominios del *track clásico* (determinista) de la competición IPC del año 2014, y viene acompañado de 20 problemas de optimalidad.

Trata acerca de actividades de excursionismo. A partir de la ubicación de las parejas, los coches y las tiendas de campaña, hay que conseguir que determinadas parejas terminen caminando juntas de un lugar a otro. Para ello, se puede armar y desarmar la tienda de campaña, conducir de un lugar a otro con una tienda de campaña desarmada y dos personas en el coche como mucho, y caminar de un lugar a otro (siempre y cuando exista una tienda de campaña en el destino que esté montada).

- Implementación del dominio:

- ⇨ **Requirements:**
  - *strips.*
  - *equality.*
  - *Typing*.
- ⇨ **Tipos**:
  - ***car***: coche.
  - ***tent***: tienda de campaña.
  - ***person***: persona.
  - ***couple***: pareja.
  - ***place***: lugar.

- ⇨ **Predicados**:
  - ***(at_tent ?x1 - tent ?x2 - place)***: localización de la tienda de campaña *x1* en el lugar *x2*.
  - ***(at_person ?x1 - person ?x2 - place)***: localización de la persona *x1* en el lugar *x2*.
  - ***(at_car ?x1 - car ?x2 - place)***: localización del coche *x1* en el lugar *x2*.
  - ***(partners ?x1 - couple ?x2 - person ?x3 - person)***: las personas *x2* y *x3* son pareja, y se identifican con *x1.* Es un predicado constante.
  - ***(up ?x1 - tent)***: la tienda de campaña *x1* está montada.
  - ***(down ?x1 - tent)***: la tienda de campaña *x1* está desmontada.
  - ***(walked ?x1 - couple ?x2 - place)***: la pareja *x1* está caminando por el lugar *x2*.
  - ***(next ?x1 - place ?x2 - place)***: el lugar *x2* se alcanza después de visitar el lugar *x1*. Es un predicado constante.
- ⇨ **Acciones**:
  - ***put_down***: si una persona está en un lugar donde hay una tienda de campaña montada, podrá desmontarla.
  - ***put_up***: si una persona está en un lugar donde hay una tienda de campaña desmontada, podrá montarla.
  - ***drive_passenger***: si dos personas y un coche se encuentran en un lugar determinado, las personas se podrán desplazar en coche a cualquier otro lugar.
  - ***drive***: si una persona y un coche se encuentran en un lugar determinado, la persona se podrá desplazar en coche a cualquier otro lugar.
  - ***drive_tent***: si una persona, una tienda de campaña desmontada y un coche se encuentran en un lugar determinado, la persona se podrá desplazar a cualquier otro lugar en coche con la tienda de campaña.
  - ***drive_tent_passenger***: si dos personas, una tienda de campaña desmontada y un coche se encuentran en un lugar determinado, las personas junto con la tienda de campaña se podrán desplazar en coche a cualquier otro lugar.
  - ***walk_together***: si dos personas que son pareja se encuentran visitando un lugar *x1*, podrán pasar a visitar otro lugar *x2* siempre y cuando este se pueda alcanzar después de visitar *x1* y siempre y cuando haya una tienda de campaña montada en *x2*.

- Implementación de los problemas:

- ⇨ **Objetos**.
  - ***car***: coches del problema.
  - ***tent***: tiendas de campaña del problema.
  - ***couple***: número de parejas en el problema.
  - ***place***: lugares posibles en el problema.

- ***person***: personas existentes en el problema.

⇨ **Predicados iniciales**:

- ***partners***: todas las parejas existentes en el problema.
- ***at_person***: la localización inicial de cada una de las personas.
- ***walked***: la localización inicial de cada una de las parejas del problema.
- ***at_tent***: la localización inicial de cada una de las tiendas de campaña.
- ***up*** o ***down***: el estado inicial de cada una de las tiendas de campaña, es decir, montada o desmontada.
- ***at_car***: la localización inicial de cada uno de los coches.
- ***next***: la continuidad establecida de los lugares.

⇨ **Predicados meta**:

- ***walked***: la localización final de cada una de las parejas del problema.

2.6.4 Dominio *Spider*

Este dominio pertenece a los dominios del *track clásico* (determinista) de la competición IPC del año 2018, y viene acompañado de 20 problemas de optimalidad.

Trata acerca del juego del solitario *Spider* pero una versión más sencilla y reducida. El objetivo del juego es descartar todas las cartas del tablero. Inicialmente, se tienen cartas distribuidas boca arriba en distintas columnas. Estas cartas se pueden mover individualmente si no tienen una carta encima de una columna a otra siempre y cuando caigan encima de una columna vacía o encima de una carta con una unidad superior (por ejemplo, la sota sobre la reina, o el as sobre el 2). No solo se pueden desplazar cartas de forma individual, también puede hacerse en bloque, siempre y cuando estas cartas estén ordenadas de mayor a menor y sean del mismo palo. Las cartas se descartarán cuando haya un mismo palo ordenado de mayor a menor. Además, se dispone de un conjunto de cartas a repartir. Siempre y cuando haya cartas encima de todas las columnas, se pueden repartir estas cartas colocando cada una de ellas encima de la última carta de cada columna, cumpliendo o no con el criterio de colocación. Se puede realizar este reparto tantas veces como sea posible, hasta que se acaben las cartas ahí ubicadas, lo cual es necesario para terminar la partida con victoria.

- Implementación del dominio:

⇨ ***Requirements***:

- *typing.*
- *conditional-effects.*
- *action-costs.*
- *negative-preconditions.*

- ⇨ **Tipos**:
  - ***cardposition***: posición de una carta. Puede ser en el tablero de juego, en el mazo de repartir o en el mazo de cartas descartadas.
  - ***card_or_tableau***: tablero de juego. Alberga las posiciones de la pila y las cartas.
  - ***card***: carta.
  - ***tableau***: pila de cartas.
  - ***deal***: mazo de cartas a repartir.
- ⇨ **Constante**:
  - ***discard***: cartas descartadas.
- ⇨ **Predicados**:
  - ***(on ?c1 - card ?c2 - cardposition)***: posición de la carta *c1* en *c2*. La posición puede ser encima de una carta, encima de una pila del tablero, encima del mazo de cartas a repartir o sobre el mazo de cartas descartadas.
  - ***(clear ?c - cardposition)***: la posición *c* no tiene ninguna carta encima. Esta puede ser otra carta, la pila de cartas del tablero o el mazo de *deal*.
  - ***(in-play ?c - card)***: la carta *c* se encuentra en el tablero de juego, es decir, no está en el mazo de *deal* ni en el de cartas descartadas.
  - ***(current-deal ?d - deal)***: indica el mazo de *deal* actual con el que se está interactuando.
  - ***(CAN-CONTINUE-GROUP ?c1 - card ?c2 - cardposition)***: indica el orden que deben seguir las cartas para poder descartarlas. Es un predicado constante.
  - **(CAN-BE-PLACED-ON ?c1 - card ?c2 - card)**: indica el orden que deben seguir las cartas para poder moverlas encima de otra carta. Es un predicado constante.
  - ***(IS-ACE ?c - card)***: Indica que la carta *c* es un As. Es un predicado constante.
  - ***(IS-KING ?c - card)***: Indica que la carta *c* es un Rey. Es un predicado constante.
  - ***(NEXT-DEAL ?d ?nd - deal)***: establece el orden que deben seguir los mazo de *deal* cuando haya que repartir sus cartas. Es un predicado constante.
  - ***(TO-DEAL ?c - card ?p - tableau ?d - deal ?next - cardposition):*** establece encima de qué pila van a acabar dichas cartas cuando haya que repartirlas de cada uno de los mazo de *deal*. Es un predicado constante.
  - ***(currently-dealing)***: indica si se está realizando en este momento un reparto de cartas de *deal* sobre cada una de las pilas de cartas del tablero.
  - ***(currently-collecting-deck)***: indica si se está realizando en este momento un descarte de las cartas ordenadas en una pila, depositándolas en el mazo de cartas descartadas.
  - ***(collect-card ?c - cardposition)***: indica qué carta se está depositando en el mazo de cartas descartadas.

- ***(part-of-tableau ?c - cardposition ?t - tableau)***: indica si el elemento *c* forma parte del tablero de juego y si se encuentra en algún lugar de la pila *t*.
- ***(movable ?c - card)***: indica si la carta puede moverse. Esto puede suceder siempre y cuando se encuentre en una pila y no tenga ninguna carta debajo, o cuando esté sobre una pila y todas las cartas puestas sobre esta estén ordenadas de acuerdo con el criterio establecido por *CAN-CONTINUE-GROUP*.
- ***(currently-updating-unmovable)***: *flag* para saber cuándo no se pueden mover en bloque las cartas del tablero, actualizando todas las cartas de dicho bloque.
- ***(make-unmovable ?c - card)***: indica que la carta *c* no podrá moverse.
- ***(currently-updating-movable)***: *flag* para saber cuándo se pueden mover cartas del tablero.
- ***(make-movable ?c - cardposition)***: indica que la carta *c* podrá moverse.
- ***(currently-updating-part-of-tableau)***: *flag* para indicar que se debe actualizar el estado del tablero, ya que algo de este ha cambiado.
- ***(make-part-of-tableau ?c - card ?t - tableau)***: *flag* para indicar que la carta *c* debe formar parte del tablero *t*.

⇨ **Funciones**:

- ***total-cost***: coste total. Es un número entero.

⇨ **Acciones**:

- ***start-dealing***: cuando no se puede descartar ninguna carta ni actualizar el tablero, se procede a repartir las cartas de *deal*. El coste de esta acción incrementa en una unidad el coste total.
- ***deal-card***: cuando se ha decidido repartir las cartas de *deal*, dependiendo del mazo *deal* actual desde donde se estén repartiendo las cartas, se cogerá dicha carta y se situará encima de la última carta de la pila correspondiente. Si la nueva carta no continua su grupo con la carta sobre la que se ha colocado, no se podrán mover en bloque, teniendo que actualizar dicho bloque.
- ***finish-dealing***: cuando se ha decidido repartir las cartas de *deal* y ya no queden más cartas en ese mazo *deal*, se dejará de repartir dichas cartas, pasando al siguiente *deal* para cuando sea necesario volver a repartirlas.
- ***move-to-card***: cuando una carta o un conjunto de cartas pueden moverse y situarse encima de la carta de otra pila, se lleva a cabo y se procede a actualizar el tablero. Si tras este movimiento, el conjunto de cartas no sigue el orden del grupo, hay que proceder a actualizar el bloque de cartas para hacerlo inmovible. Por otra parte, si la carta utilizada para mover el bloque no continuaba el grupo con su posición de origen, hay que proceder a hacer la pila de origen movible (hasta donde se pueda). El coste de esta acción incrementa en una unidad el coste total.

- ***move-to-tableau***: cuando una carta o un conjunto de cartas pueden moverse y situarse encima de otra pila vacía, se lleva a cabo y se procede a actualizar el tablero. Si tras este movimiento, la carta utilizada para mover el bloque no continuaba el grupo con su posición de origen, hay que proceder a hacer la pila de origen movible (hasta donde se pueda). El coste de esta acción incrementa en una unidad el coste total.
- ***change-tableau-and-continue***: cuando se decide actualizar el tablero y no se estén descartando cartas, se cambia la localización de una de las cartas recién movida, indicando sobre qué pila está esta carta. Tras esto, se identifica la carta que está por debajo de la pila y se marca para poder actualizar posteriormente la situación de esta carta.
- ***change-tableau-and-stop***: cuando se decide actualizar el tablero y no se estén descartando cartas, se cambia la localización de una de las cartas recién movida, indicando sobre qué pila está esta carta. Dado que esta era la última carta del bloque movido, se indica que no es necesario seguir actualizando el tablero.
- **make-unmovable-and-continue**: cuando se decide actualizar el tablero para hacer algunas cartas inmovibles, si no se están descartando cartas ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. A pesar de que dos cartas superpuestas sigan el orden del grupo, si se ha decidido hacer inmóvil la carta inferior, se hace. Una vez hecho esto, se indica que debe hacerse inmóvil posteriormente la carta situada encima.
- ***make-unmovable-and-stop-at-tableau***: cuando se decide actualizar el tablero para hacer algunas cartas inmovibles, si no se están descartando cartas ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. Cuando la carta que se quiere hacer inmovible es la primera de la pila, se hace inmovible y se indica que no se va a actualizar más el tablero para hacer más cartas inmovibles.
- ***make-unmovable-and-stop-at-unmovable***: cuando se decide actualizar el tablero para hacer algunas cartas inmovibles, si no se están descartando cartas ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. Cuando la carta que se quiere hacer inmovible está situada sobre una que ya es inmovible, se hace inmovible y se indica que no se va a actualizar más el tablero para hacer más cartas inmovibles.
- ***make-movable-and-continue***: cuando se decide actualizar el tablero para hacer algunas cartas movibles, si no se están descartando cartas, ni haciendo cartas inmovibles, ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. Cuando dos cartas superpuestas siguen el orden del grupo, si se ha decidido hacer inmóvil la carta inferior, se hace. Una

vez hecho esto, se indica que debe hacerse móvil posteriormente la carta situada encima.

- ***make-movable-ignore-pile***: cuando se decide actualizar el tablero para hacer algunas cartas movibles, si no se están descartando cartas, ni haciendo cartas inmovibles, ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. Cuando se indica que debe hacerse movible una pila, se ignora y se indica que no se va a actualizar más el tablero para hacer más cartas movibles.
- ***make-movable-and-stop***: cuando se decide actualizar el tablero para hacer algunas cartas movibles, si no se están descartando cartas, ni haciendo cartas inmovibles, ni actualizando el tablero para cambiar la situación de las cartas (indicando a qué pila pertenecen), podría ejecutarse esta acción. Cuando dos cartas superpuestas no siguen el orden del grupo, si se ha decidido hacer inmóvil la carta inferior, se hace. Una vez hecho esto, se indica que no se va a actualizar más el tablero para hacer más cartas movibles.
- ***start-collecting-deck***: cuando no se puede actualizar nada del tablero, ni se están repartiendo las cartas de *deal*, si el as se encuentra en una pila y no hay ninguna carta puesta encima de ella, puede procederse a descartar un bloque de cartas, comenzando por el as. El coste de esta acción incrementa en una unidad el coste total. En principio solo se pueden coleccionar cartas cuando se ha completado el grupo, pero la operación se completará ya que la única forma de salir del estado de descarte es terminar descartando un rey.
- ***collect-card***: cuando se ha decidido descartar cartas, se descarta la carta que se ha seleccionado. Además, se marca la carta inmediatamente superior para descartarla en un futuro, siempre y cuando formen parte del mismo grupo. Además, la carta superior pasará a ser *clear*.
- ***finish-collecting-deck***: cuando se ha decidido descartar cartas, se descarta la carta que se ha seleccionado. Además, si dicha carta es un rey, se para el descarte, ya que se supone que se ha descartado todo el grupo. Además, la carta superior pasará a ser *clear*, y se indicará que deben actualizarse las cartas de dicha pila para hacerlas movibles, empezando con hacer movible la que está inmediatamente superior.

- Implementación de los problemas:

⇨ **Objetos**.

- ***card***: todas las cartas de la partida. Siguen la siguiente nomenclatura: *dx-sy-vz*, siendo *x*, *y* y *z* valores numéricos. La *v* se corresponde con el número de la carta, la *s* con el color de la carta, y la *d* con el número de baraja.

- ***tableau***: una por cada pila.
- ***deal***: una por cada mazo de *deal*.

⇨ **Predicados iniciales**:

- ***on***: posición inicial de cada una de las cartas de la partida.
- ***clear***: con todas las cartas que no tienen ninguna otra carta debajo.
- ***part-of-tableau***: con todas las cartas y pilas indicando en qué pila está cada una de ellas.
- ***movable***: con todas las cartas que pueden moverse.
- ***in-play***: con todas las cartas que están sobre el tablero.
- ***current-deal***: con el primer mazo de *deal* que se repartirá.
- ***CAN-CONTINUE-GROUP***: con toda la secuencia de cartas para poder descartarlas. Como norma general, una carta seguirá el mismo grupo siempre y cuando se deposite encima de otra carta del mismo color y un valor numérico una unidad superior.
- ***CAN-BE-PLACED-ON***: Como norma general, una carta podrá moverse encima de otra siempre y cuando se deposite encima de otra carta con un valor numérico una unidad superior.
- ***IS-ACE***: con todas las cartas que son ases (*v0*).
- ***IS-KING***: con todas las cartas que son reyes (*vx*, siendo *x* el mayor valor numérico que acompaña a *v*).
- ***NEXT-DEAL***: con el orden de secuencia de mazos de *deal*.
- ***TO-DEAL***: con el orden de reparto de las cartas de *deal* sobre su correspondiente pila.

⇨ **Predicados meta**:

- ***on***: de todas las cartas de la partida. Deben estar todas descartadas.
- ***clear***: de las pilas y de los mazos *deal*. Realmente, que todas las cartas están descartadas implica que sobre las pilas y sobre los mazos de *deal* no haya ninguna carta. Sin embargo, que haya predicados redundantes en las metas ayuda en la búsqueda.

⇨ **Objetivo**:

- **Minimizar *total-cost***. Empieza valiendo 0.

2.6.5 Dominio *Tetris*

El dominio *Tetris* pertenece a los dominios del *track clásico* (determinista) de la competición IPC del año 2014, y viene acompañado de 17 problemas de optimalidad.

Trata acerca del juego del *Tetris*, pero una versión distinta. Esta versión opera con tres tipos distintos de bloque: cuadrado de una posición, una línea de dos posiciones o una *L* que ocupa 3 posiciones. Estos bloques se sitúan todos en la parte superior del tablero, y el objetivo es desplazarlos individualmente hasta colocarlos en las primeras filas del

tablero, dejando vacío su lugar original. Como resulta evidente, no pueden moverse de cualquier manera. Solo pueden desplazarse una unidad a la izquierda, derecha, arriba o abajo siempre y cuando no haya ninguna pieza que entorpezca el movimiento, ya que no se pueden superponer.

- Implementación del dominio:

- ⇨ ***Requirements***
  - *typing.*
  - *equality.*
  - *negative-preconditions.*
  - *action-costs.*
- ⇨ **Tipos**:
  - ***one_square***: pieza del *Tetris*. Se basa en un cuadrado *1X1*.
  - ***two_straight***: pieza del *Tetris*. Se basa en una línea *2X1* o *1X2*.
  - ***right_l***: pieza del Tetris. Se basa en una *L* ubicada sobre tres posiciones.
  - ***position***: posición.
- ⇨ **Predicados**:
  - ***(clear ?xy - position)***: sobre la posición *xy* no se encuentra ninguna pieza.
  - ***(connected ?x - position ?y - position)***: adyacencia entre las dos casillas *x* e *y*. Es un predicado constante.
  - ***(at_square ?element - one_square ?xy - position)***: posición del cuadrado *element*.
  - ***(at_two ?element - two_straight ?xy - position ?xy2 - position)***: posición de la línea *element*.
  - ***(at_right_l ?element - right_l ?xy - position ?xy2 - position ?xy3 - position)***: posición de la pieza en forma de *L* denominada *element*.
- ⇨ **Funciones**:
  - ***total-cost***: coste total. Es un número entero.
- ⇨ **Acciones**:
  - ***move_square***: un cuadrado podrá moverse a una casilla adyacente siempre y cuando esta no esté ocupada por otra pieza. Cuando esto suceda, la posición de origen se liberará, el cuadrado cambiará de posición y esta nueva posición dejará de estar libre. El coste de esta acción incrementa en una unidad el coste total.
  - ***move_two***: cuando una pieza lineal se sitúa en el tablero, un extremo se considera la *cola* y otro la *cabeza*. Cuando se identifica una posición adyacente a la *cabeza* de la pieza siempre y cuando no esté ocupada, esta pasará a ser la *cabeza* de la pieza (posición no libre), la antigua *cabeza* se convertirá en la *cola* (posición no libre), y la *cola* se liberará (posición libre). Esto simulará los desplazamientos y las rotaciones de la pieza, aunque su comportamiento está limitado por la distinción entre *cola* y *cabeza*. El coste de esta acción incrementa en dos unidades el coste total.

- ***move_l_right***: cuando la pieza *L* se mueve hacia la derecha, se liberan dos posiciones, pero se ocupan otras dos. Para que la acción sea posible, estas dos nuevas posiciones deben estar inicialmente libres. El coste de esta acción incrementa en tres unidades el coste total.
- ***move_l_left***: cuando la pieza *L* se mueve hacia la izquierda, se liberan dos posiciones, pero se ocupan otras dos. Para que la acción sea posible, estas dos nuevas posiciones deben estar inicialmente libres. El coste de esta acción incrementa en tres unidades el coste total.
- ***move_l_up***: cuando la pieza *L* se mueve hacia arriba, se liberan dos posiciones, pero se ocupan otras dos. Para que la acción sea posible, estas dos nuevas posiciones deben estar inicialmente libres. El coste de esta acción incrementa en tres unidades el coste total.
- ***move_l_down***: cuando la pieza *L* se mueve hacia abajo, se liberan dos posiciones, pero se ocupan otras dos. Para que la acción sea posible, estas dos nuevas posiciones deben estar inicialmente libres. El coste de esta acción incrementa en tres unidades el coste total.

- Implementación de los problemas:

⇨ **Objetos**.
- ***position***: todas las posiciones del tablero de juego.
- ***one_square***: posiciones iniciales de los cuadrados.
- ***two_straight***: posiciones iniciales de las piezas lineales.
- ***right_l***: posiciones iniciales de las piezas con forma de *L*.

⇨ **Predicados iniciales**:
- ***connected***: con todas las posiciones, creando la estructura del tablero de juego.
- ***clear***: con todas las posiciones donde no hay una pieza.
- ***at_right_l***: la localización inicial de cada una de las piezas en forma de *L*. Se sitúan inicialmente en la parte superior del tablero.
- ***at_two***: la localización inicial de cada una de las piezas lineales. Se sitúan inicialmente en la parte superior del tablero.
- ***at_square***: la localización inicial de cada una de las piezas cuadradas. Se sitúan inicialmente en la parte superior del tablero.

⇨ **Predicados meta**:
- ***clear***: de las posiciones superiores. Dado que el problema comienza con las piezas situadas en la parte superior del tablero, el objetivo final es acomodar las piezas en la parte inferior, es decir, liberando las posiciones superiores.

⇨ **Objetivo**:
- **Minimizar *total-cost***. Empieza valiendo 0.

2.7 Trabajos relacionados

Antes de comenzar con la investigación, es aconsejable partir de trabajos similares, para no empezar desde cero en la medida de lo posible. A pesar de que el avance de la inteligencia artificial y de sus estudios está bastante en auge, se han recopilado artículos no necesariamente de los últimos años. De esta forma, mediante diversas fuentes de información distanciadas en el tiempo, se tomará una decisión en los puntos 2.7.4 Comparación entre los trabajos relacionados y su implicación en el proyecto y 2.9 propuesta.

Se han seleccionado tres artículos. Se comentarán a continuación muy brevemente para no pecar de plagio, extrayendo tan solo aquella información útil para el proyecto a realizar.

2.7.1 *iPDB*

iPDB es un algoritmo independiente del dominio del que ya se habó anteriormente en el punto 2.5.2 iPDB. Se vuelve a hacer referencia a este ya que es relevante destacarlo como trabajo relacionado (Bonet, Patrik, Botea, Helmert, & Koening, 2007). Esta relación se debe a que este algoritmo, entre otras cosas, construye PDBs, exactamente lo que se quiere conseguir, pero mediante un proceso de búsqueda. Este proceso de búsqueda puede acarrear un gran tiempo de ejecución.

2.7.2 *Merge and Shrink*

Merge and Shrink (Sievers, Wehrle, & Helmert, An Analysis of Merge Strategies for Merge-and-Shrink Heuristics, 2016) (Sievers, Wehrle, & Helmert, Generalized Label Reduction for Merge-and-Shrink Heuristics, 2014) es otro algoritmo de planificación independiente del dominio. La idea principal de este algoritmo es la de reducción de etiquetas, que es una técnica destinada a simplificar las familias de los sistemas de transición etiquetados, eliminando distinciones entre algunas de dichas etiquetas de transición cuando son semánticamente equivalentes. Entrar en detalle en este algoritmo no es relevante para el actual proyecto. Sin embargo, aunque no hace uso de PDBs, crea estados abstractos mediante árboles binarios lineales y no lineales en su proceso de *Merge*. Este algoritmo suele acarrear también bastante tiempo de ejecución, a costa de poder resolver los problemas de planificación con una buena heurística.

2.7.3 Bases de datos de patrones simbólicas

Las bases de datos de patrones simbólicas (Edelkamp, 2002), más conocidas por sus siglas SPDB, sirven como estimaciones heurísticas, produciendo bases de datos de patrones mucho más grandes y precisas que las generadas con métodos explícitos.

El artículo donde desarrollan por primera vez las SPDBs hace referencia tanto a las PDBs como a los diagramas binarios de decisión (BDDs). Estas últimas se basan en la construcción de árboles donde en cada nivel se preguntan acerca de si se cumplen una cierta instancia en el estado actual, dando una respuesta binaria que conduce a una rama del árbol o a la otra. Se plantean cuestiones acerca de cómo ordenar los nodos del árbol, pues los resultados son distintos. Existen evidencias de que las BDDs guardan un cierto parecido con las PDBs, pudiendo realizar en ocasiones conversiones de una a la otra. Por otra parte, una de las conclusiones del estudio es que las PDBs son una buena alternativa para mejorar la heurística.

2.7.4 Comparación entre los trabajos relacionados y su implicación en el proyecto

iPDB es un buen algoritmo para construir buenas PDBs, pero conlleva mucho tiempo. A su vez, *Merge and Shrink* también obtiene buenas heurísticas, a costa de un gran tiempo de ejecución. Por otra parte, hay constancia de que las PDBs son una gran alternativa en el desarrollo de heurísticas. Estas ideas son suficientes para dar explicación a la idea del proyecto, pues es difícil encontrar estudios que se ajusten al proceso de investigación que se va a seguir y al propósito de este. Las ideas que aquí se proponen no se encuentran en ningún otro estudio.

De forma general, hoy en día el desarrollo de una buena heurística conlleva un gran consumo de tiempo, y más en problemas complejos. Las PDBs son una buena herramienta, pero también tienen este problema al aplicarlas actualmente. Por otra parte, también existen evidencias de que el orden de colocación de las variables es relevante (como en las BDDs), al igual que su presencia en ciertas PDBs, lo que implica que seleccionar variables es determinante para obtener una buena heurística informada, pues no todas tienen la misma relevancia. Dicho esto, y enlazando con la crítica al estado del arte 2.8 Crítica al estado del arte y con la propuesta planteada 2.9 Propuesta, para mejorar los resultados de los algoritmos de planificación de hoy en día sería una buena opción plantear un estudio de las variables *relevantes* en las PDBs obtenidas por diversos algoritmos en algunos dominios, y establecer una relación entre todas ellas. De esta forma, no se dará la necesidad de implementar un algoritmo de búsqueda que seleccione las variables y cree las PDBs.

2.8 Crítica al estado del arte

Dentro del ámbito de la planificación automática se pueden manejar diversos algoritmos resolutivos, pero consumen demasiado tiempo. Por otra parte, la potencia de estos podría ser mejorable. Es necesario buscar otras alternativas más adecuadas mediante diversos procesos de investigación.

En concreto, existe un tipo de heurísticas llamadas PDB, tal y como se ha explicado anteriormente, las cuales son usadas por diversos planificadores, como *Fast Downward*, y proporcionan buenos resultados. Existen algoritmos que hacen uso de estas, como *iPDB*, pero suelen requerir de un tiempo considerable para poder obtener un patrón lo suficientemente acertado como para poder resolver el problema. En otras ocasiones, incluso, no es capaz de resolverlos, aun ignorando el tiempo de ejecución y la cantidad de memoria utilizada. Esto se debe, principalmente, a que hacen uso de algoritmos de búsqueda para poder obtener estas PDBs.

Hay constancia de que se han realizado investigaciones para obtener heurísticas mediante otras alternativas, como en *Merge and Shrink*, llegando a buenas conclusiones, aunque, igualmente, siguen requiriendo mucho tiempo de cómputo. Es posible que, indagando más en analizar las PDBs, se pueda llegar a obtener mejores resultados a los que se obtienen en la actualidad.

2.9 Propuesta

Tal y como se ha comentado en la introducción Sección 1 Introducción y objetivos, y de acuerdo con la crítica al estado del arte 2.8 Crítica al estado del arte, se propone investigar las variables implicadas en las PDBs obtenidas por algoritmos de Fast Downward que hagan uso de estas, es decir, *Canonical PDB*, *iPDB* y *Zero-One PDB*. Hacer esto con cada uno de los 5 dominios propuestos y generalizar para identificar qué variables son relevantes en cada dominio, y cómo agruparlas adecuadamente para obtener PDBs que sean capaces de resolver más problemas en el menor tiempo posible y haciendo uso de la menor cantidad de memoria posible. En el peor de los casos tan solo se pretenderá reducir el tiempo de ejecución al suprimir el proceso de búsqueda de las PDBs.

Una vez hecho esto, estudiar cómo obtener dichas variables dado un dominio cualquiera, sin tener la necesidad de indagar en el dominio en cuestión. Lo que se propone en esta última parte del estudio, no solo es implementar un algoritmo en función de las observaciones contempladas, sino también abrir una puerta para investigaciones futuras que hagan uso de las deducciones y datos recabados en este proyecto. Se probarán diversas alternativas y se asumirán otras, siempre dejando

planteadas futuras líneas de investigación que podrían aportar, incluso, mejores resultados y conclusiones que las presentes en este documento.

SECCIÓN 3. DISEÑO DE LA SOLUCIÓN

En este punto se establecerá la metodología de estudio, posibles alternativas y cuestiones acerca de la implementación de los algoritmos, como los requisitos o la arquitectura del sistema.

Dado que muchos pasos de la metodología, de las posibles alternativas y del código implementado dependen de los resultados obtenidos en la investigación, se adelantarán algunos resultados.

3.1 Metodología de estudio

A continuación, se establece la metodología de estudio. La metodología se adapta a las conclusiones extraídas de la investigación, pero su esqueleto es fijo y se acomoda a los objetivos del proyecto.

3.1.1 Esqueleto de la investigación

A pesar de que la investigación se moldea a los resultados obtenidos durante la experimentación, su esqueleto se definió previamente. Esta consta de los siguientes pasos:

1) **Establecimiento de los objetivos clave de la investigación**: aunque inicialmente ya se establecieron los objetivos del proyecto, es necesario recalcar e interiorizar aquellos objetivos referidos a la investigación per se, con el fin de establecer una adecuada metodología de estudio.
2) **Cuestiones generales**: antes de comenzar con el proceso de experimentación, es necesario concretar ciertos aspectos generales del proyecto, como el lenguaje de programación, el sistema operativo, etc.
3) **Elección de los dominios**: elección de los dominios que se van a estudiar para realizar una heurística dependiente del dominio por cada uno de ellos.
4) **Elección de los algoritmos de estudio**: analizar los algoritmos relacionados con PDBs que *Fast Downward* ofrece. Establecer cuáles de ellos usar para obtener las variables *relevantes* e implementar el algoritmo, y cuáles de ellos usar para las pruebas y la evaluación del algoritmo implementado.
5) **Ejecución de los algoritmos y observación de resultados**: ejecutar los algoritmos anteriormente seleccionados y observar qué variables son relevantes y cómo conforman las PDBs. Hacer esto para cada uno de los dominios elegidos en el paso 3.

6) **Implementación y ejecución del algoritmo dependiente del dominio**: implementar y ejecutar un algoritmo que se ajuste a las observaciones realizadas en el paso anterior. Hacer esto para cada uno de los dominios seleccionados.
7) **Pruebas del algoritmo dependiente del dominio**: verificar su correcto funcionamiento y validarlo con los requisitos de usuario.
8) **Evaluación del algoritmo (dependiente del dominio)**: evaluar el algoritmo implementado en el paso 6 comparándolo con otros sobre PDBs. Hacer esto para cada uno de los dominios seleccionados.
9) **Extracción de conclusiones del dominio**: extraer conclusiones en función de la evaluación realizada en el paso anterior. Hacer esto para cada uno de los dominios seleccionados.
10) **Observación de características similares entre las PDBs relevantes de cada dominio**: se estudiará que características comparten todas las variables seleccionadas como relevantes durante la investigación entre todos los dominios, en función de su posición en la codificación de estos.
11) **Implementación y/o modificación y ejecución del algoritmo independiente del dominio**: implementar un algoritmo que haga uso de las observaciones realizadas en el anterior paso, modificando otros algoritmos en caso de ser necesario. Ejecutar finalmente todos los algoritmos implementados y/o modificados.
12) **Pruebas del algoritmo independiente del dominio**: verificar su correcto funcionamiento y validarlo con los requisitos de usuario.
13) **Evaluación del algoritmo (independiente del dominio)**: evaluar el algoritmo implementado en el paso 11 comparándolo con otros sobre PDBs.
14) **Extracción de conclusiones del estudio independiente del dominio**: extraer conclusiones en función de la evaluación realizada en el paso anterior.
15) **Establecimiento de posibles trabajos futuros**: en función de las conclusiones extraídas durante la realización de este proyecto, establecer posibles futuras líneas de estudio que podrían proporcionar buenos resultados, con el fin de progresar la investigación aquí iniciada. De igual forma, se proponen otras líneas de investigación durante el transcurso del proyecto, al tomar decisiones que encauzan la experimentación.

3.1.2 Establecimiento de los objetivos clave de la investigación

El objetivo clave de la investigación, tal y como se ha comentado en la Sección 1 Introducción y objetivos y de acuerdo con el título del proyecto, es crear heurísticas dependientes del dominio, y posteriormente generalizar y crear una independiente del dominio, todas mediante PDBs.

El estudio se divide, por tanto, en dos partes fundamentales. A su vez, cada una de dichas partes se divide en 5 pasos base: observación, implementación, verificación y validación, evaluación y conclusiones. Los pasos de la estructura básica del punto 3.1.1 Esqueleto de la investigación los contiene todos. Los primeros 4 pasos establecen la metodología básica, cuestiones generales, dominios y algoritmos a utilizar; los siguientes 5 pasos corresponden al estudio dependiente del dominio; los 5 siguientes a la heurística independiente del dominio; y el último establece líneas de investigación futuras.

3.1.3 Cuestiones generales

Además de los pasos a seguir, se tendrán en consideración algunas cuestiones generales:

- **Tipos de heurísticas a desarrollar**: se van a desarrollar heurísticas dependientes del dominio, y una final independiente del dominio en función de los resultados de estas primeras heurísticas.

- **Exhaustividad de la experimentación**: la exhaustividad de la investigación se ha ajustado de acuerdo con lo que se espera en un Trabajo Fin de Grado. El número de dominios elegidos es de 5, y la heurística independiente a implementar, una.

El proceso de investigación para cada dominio se compone de las siguientes fases fundamentales: ejecución de algoritmos, observación, implementación del nuevo algoritmo, ejecución del nuevo algoritmo, verificación y validación del algoritmo, evaluación y conclusiones. Para la heurística independiente del dominio, dado que no es necesario volver a ejecutar ciertos algoritmos nuevamente, las fases principales son las siguientes: observación, implementación del nuevo algoritmo, ejecución del nuevo algoritmo, verificación y validación, evaluación y conclusiones.

- **Elección del planificador**: el planificador seleccionado es *Fast Downward* dado que ofrece una gran variedad de algoritmos, es sencillo de usar y de modificar, y ofrece la documentación necesaria para comprenderlo.

- **Elección del sistema operativo**: *Fast Downward* funciona en Linux, Windows y en Mac OS X. Sin embargo, aconsejan su uso en Linux. Por esa razón, se ha decidido trabajar en Ubuntu.

- **Lenguajes de programación y planificación**: dado que *Fast Downward* está implementado tanto en *Python* como en C++, es necesario aprender ambos lenguajes. Sin embargo, conocía *Python* anteriormente, y es con el que me siento más cómodo. Su uso me parece muy intuitivo y sencillo, por lo que los *Scripts* se implementarán mediante

este lenguaje de programación. Además, es necesario comprender PDDL, pues los dominios y sus problemas están implementados en este lenguaje.

- **Herramientas auxiliares**: aparte de *Fast Downward*, se requiere usar más herramientas durante la investigación. Estos son: editores de texto como *Gedit*, un navegador de internet como *Mozilla Firefox* o algún lector de PDFs como *Adobe Acrobat Reader*.

3.1.4 Elección de los dominios

Para ajustarse dentro de los límites de lo que se espera en un Trabajo Fin de Grado, el número de dominios que se van a analizar es 5. Estos dominios deben cumplir 4 premisas clave:

1) Deben ser dominios bien construidos e implementados.
2) Deben ser relativamente recientes.
3) Deben ofrecer buenos resultados usando PDBs.
4) Deben ser dominios de optimalidad.

La primera premisa se puede solventar seleccionando dominios fiables, es decir, aquellos implementados para IPC, la Competición Internacional de Planificación.

El hecho de que sean recientes disminuye las posibilidades de que hayan sido ya empleados en otras investigaciones. Dicho esto, se han escogido dominios pertenecientes a las dos últimas competiciones internacionales de planificación automática (IPC). Corresponden con las de los años 2014 y 2018.

La tercera premisa es más difícil de contemplar a simple vista sin ejecutar los dominios. Para ello, se ha realizado un pequeño estudio con muchos de los dominios del *track clásico* (determinista) de IPC de los años 2014 y 2018. Ese estudio se muestra en el apartado 4.1.1 Selección de dominios, y consiste en seleccionar un conjunto de dominios e ir descartando aquellos que, aparentemente, no funcionan bien mediante PDBs. Descartar un dominio potencialmente bueno no es ningún problema, dado que bastará con seleccionar 5 dominios adecuados y dispares. Haciendo esto, se llega a la conclusión de que los dominios *Snake*, *Termes*, *Hiking*, *Spider* y *Tetris* cumplen estas 4 premisas.

IPC ofrece dominios y problemas tanto de satisfabilidad como de optimalidad. Se han escogido dominios de esta índole para la realización del proyecto por el mero hecho de orientar y enfocar la investigación, tomando ciertas decisiones. Sin embargo, el hecho de encontrar una solución óptima es más difícil que suceda por puro azar, al contrario que los problemas de satisfabilidad, en los que el coste de la solución (el plan) varía dependiendo de la semilla aleatoria, las PDBs elegidas, etc. Futuras investigaciones

podrían probar con dominios y problemas de satisfabilidad y compararlos con los de optimalidad.

3.1.5 Elección de los algoritmos de estudio

Se deben seleccionar algoritmos tanto para crear las PDBs en los procesos de observación y evaluación, procesos citados en el punto 3.1.2 Establecimiento de los objetivos clave de la investigación, como para ejecutar el planificador en busca del plan solución.

La elección del algoritmo de optimalidad que debe resolver el planificador ha sido directa. *A** forma parte de los algoritmos de búsqueda heurística más estudiados durante la carrera. Además, ofrece resultados óptimos. Por esa razón se seleccionó.

No se considera necesario utilizar otros algoritmos de planificación, pues el objetivo del proyecto no es evaluar estos algoritmos. Durante los procesos de investigación y experimentación en ciertas ocasiones es necesario mantener ciertos valores constantes para determinar mejor la variabilidad entre resultados. De todos modos, no se afirma rotundamente que variar el algoritmo de planificación vaya a ser trabajo en vano, solo se supone que no va a ser del todo relevante para el proyecto.

La elección de los algoritmos utilizados para observar cuáles son las variables relevantes en cada dominio se ha llevado a cabo mediante un breve estudio, al igual que la elección de los algoritmos para implementar el algoritmo y para comparar y evaluar resultados. Todos estos algoritmos están relacionados con PDBs, pues es el tema del proyecto. No es del todo relevante la relación con otros algoritmos que no hagan uso de PDBs, ya que lo que se quiere conseguir con esta investigación es determinar si, en caso de que el uso de PDBs sea bueno en ciertos dominios, mejorar dichos resultados. Futuras líneas de investigación podrían realizar este tipo de comparaciones, solo que aquí no interesa, pues se trata de una investigación base que podría desencadenar futuros estudios aún más específicos.

Además, todos estos algoritmos pertenecen a *Fast Downward*, ya que ha sido el planificador que se ha elegido para trabajar. De igual forma, futuras investigaciones podrían centrarse en otros planificadores.

Aquellos algoritmos que finalmente se seleccionarán para el proceso de observación son iPDB, *Canonical PDB* y *Zero-One PDB*. Por otra parte, aquellos algoritmos seleccionados que acompañan al *script* de creación de PDBs son *Canonical PDB* y *Zero-One PDB*, pues ambos permiten introducir patrones de forma manual. Por último, aquellos algoritmos seleccionados para el proceso de evaluación serán los mismos que se han seleccionado para el proceso de investigación más el algoritmo implementado en cuestión.

3.1.6 Ejecución de los algoritmos y observación de resultados

En función de los algoritmos elegidos en el punto 4.1.2 Selección de algoritmos y los dominios seleccionados en el punto 4.1.1 Selección de dominios, se procederá a ejecutar y a observar los resultados. La ejecución de los algoritmos se llevará a cabo mediante *scripts* que automatizarán el proceso, tal y como se explica en el punto 3.3 Código implementado. Tras esto, se observarán qué variables son relevantes y cómo conforman las PDBs en cada uno de los dominios anteriormente seleccionados.

Las tablas que se muestran indican: el tiempo de ejecución del proceso de creación de las PDBs, el tiempo de poda y la asociación de PDBs aditivas, el tiempo de búsqueda, la cantidad de memoria utilizada, el número de nodos expandidos y el coste. En caso de no resolverlo, también se mostrará mediante un guión ("-"). En estos problemas sin resolver no se mostrará el número de nodos expandido, ni la memoria utilizada, ni el tiempo total hasta que el algoritmo se detiene, pues en planificación óptima y en este proyecto prevalece el número de problemas resueltos. Respecto a los decimales mostrados, aunque tan solo se requieran dos para realizar un análisis correcto de los datos, se han introducido todos los que se han obtenido tras la ejecución.

No se han incorporado gráficas porque, al no mostrar ninguna otra información que las tablas, generan cierta redundancia. Eso es debido, en parte, a que los problemas no están clasificados por dificultad, por lo que no se puede extraer ninguna conclusión respecto a su complejidad. Además, hay algunos datos que no se pueden representar, como el tiempo de búsqueda en aquellas ejecuciones en las que no se ha terminado de generar las PDBs.

Respecto a la observación, se centrará sobre todo en las PDBs de los problemas resueltos, aunque los otros también servirán de ayuda, ya que son problemas complicados que requieren de mucha memoria. Por eso no se suelen resolver, no porque la heurística obtenida sea mala. Por otra parte, no se tendrán en cuenta casos que no se repiten, pues se pretende generalizar.

Este punto se encuentra íntegro en el proceso de investigación Sección 4 Investigación.

3.1.7 Implementación y ejecución del algoritmo dependiente del dominio

En este punto, se implementará y se ejecutará un algoritmo que sea capaz de construir PDBs que se asimilen a las observadas en las ejecuciones de los mejores algoritmos del paso 3.1.6 Ejecución de los algoritmos y observación de resultados. Esto se realizará para cada uno de los dominios seleccionados. Por tanto, se determinará que el algoritmo implementado es apropiado cuando sea casi tan bueno o mejor que ese mejor

algoritmo, decisión que se tomará en el punto 3.1.9 Evaluación del algoritmo (dependiente del dominio).

Este apartado se explica tanto en el proceso de investigación Sección 4 Investigación como en los apartados 3.3.3 *Script* de algoritmo para *Snake*, 3.3.4 *Script* de algoritmo para *Termes*, 3.3.5 *Script* de algoritmo para *Hiking*, 3.3.6 *Script* de algoritmo para *Spider* y 3.3.7 *Script* de algoritmo para *Tetris*.

3.1.8 Pruebas del algoritmo dependiente del dominio

Las pruebas del algoritmo consisten en seleccionar 3 problemas distintos (esta sería la batería de pruebas unitarias de *software*), y contrastar la salida esperada con la obtenida. Es mejor seleccionar una batería de pruebas que evalúe las diferentes situaciones que escoger numerosos problemas que evalúen lo mismo y que no aporten nada distinto, pues la mayor parte de los problemas son muy similares entre sí. De esta forma, los tests se realizarán sobre el algoritmo teniendo en cuenta las características del problema introducido. Una vez verificado, la validación simplemente consistiría en observar si los requisitos de usuario coinciden con lo que el algoritmo ofrece, que son solo dos.

Estas pruebas se encuentran en el punto Sección 4 Investigación.

3.1.9 Evaluación del algoritmo (dependiente del dominio)

Se procede a comparar los algoritmos del punto 3.1.5 Elección de los algoritmos de estudio con el algoritmo implementado, dependiendo del dominio en cuestión. Como resulta lógico, es necesario ejecutar previamente dicho algoritmo. Tal y como se ha comentado anteriormente, se determinará que el algoritmo implementado es apropiado cuando sea casi tan bueno o mejor que el algoritmo que proporcione los mejores resultados.

De nuevo, la ejecución de estos algoritmos se llevará a cabo mediante *scripts* que automatizarán el proceso, tal y como se explica en el punto 3.3.10 *Scripts* para automatizar las llamadas a algoritmos. Sin embargo, muchos de estos algoritmos ya se ejecutaron en el punto de observación 3.1.6 Ejecución de los algoritmos y observación de resultados. Este punto se encuentra íntegro en el proceso de investigación Sección 4 Investigación.

3.1.10 Extracción de conclusiones del dominio

En este paso simplemente se extraerán las conclusiones oportunas de cada dominio, en función del proceso de investigación que se ha realizado en torno a él. Este punto se desarrolla en la sección Sección 4 Investigación. Se acepatará el algoritmo como válido si alcanza o supera al mejor algoritmo analizado de *Fast Downward* que lo resuelva.

3.1.11 Observación de características similares entre las PDBs relevantes de cada dominio

Sin necesidad de ejecutar nada adicional, se compararán las características similares que existen entre las variables identificadas por los algoritmos implementados en cada uno de los dominios. Para ello, dado que el futuro algoritmo a implementar no comprenderá la semántica, se identificarán dichas similitudes mediante la ubicación de cada una de estas variables *relevantes* en la codificación de su dominio correspondiente, basándose también en la localización de las variables meta. Bajo esta condición inicial, se irá construyendo la metodología a través de la observación y el análisis de diferentes resultados.

Posteriormente, se estudiarán distintas alternativas para construir PDBs mediante estas variables y se seleccionará una. Entre estas alternativas se baraja la posibilidad de implementar un nuevo algoritmo o de modificar uno existente.

Este punto se encuentra íntegro en el proceso de investigación Sección 4 Investigación.

3.1.12 Implementación y/o modificación y ejecución de algoritmos independientes del dominio y ejecución

En función de las observaciones realizadas en el 4.7.1 Observación de características similares entre las PDBs relevantes de cada dominio, se construirá un algoritmo que sea capaz de identificar esas variables relevantes sin estudiar el dominio en cuestión. Además, mediante la modificación de un algoritmo o la implementación de otro distinto, se construirán las PDBs en función de estas variables *relevantes*. Estos algoritmos no deben hacer distinción entre dominios, es decir, no debe de ejecutarse de forma distinta dependiendo del dominio introducido. Por tanto, tras su ejecución, se determinará que la heurística independiente del dominio es adecuada cuando proporcione resultados casi tan buenos o mejores que el mejor algoritmo seleccionado, decisión que se tomará en el punto 4.7.5 Extracción de conclusiones.

Este apartado se explica tanto en el proceso de investigación Sección 4 Investigación como en los apartados 3.3.8 *Script* de algoritmo independiente del dominio y 3.3.9 Modificaciones en iPDB. Por otra parte, tras la ejecución, se mostrará la información en tablas, tal y como se ha comentado en el punto 3.1.6 Ejecución de los algoritmos y observación de resultados.

3.1.13 Pruebas del algoritmo independiente del dominio

Las pruebas del algoritmo consiste en seleccionar 1 problema por dominio (esta sería la batería de pruebas unitarias de *software*), y contrastar la salida esperada y la obtenida. Es mejor seleccionar una batería de pruebas que evalúe diferentes situaciones que seleccionar numerosos problemas que evalúen lo mismo y no aporten nada distinto. El algoritmo implementado actúa de la misma forma con todos los problemas de todos los dominios, por lo que probar 5 problemas (uno por dominio) es suficiente, ya que se parsea el dominio. De esta forma, los tests se realizarán sobre el algoritmo teniendo en cuenta las características del dominio introducido. Una vez verificado, la validación simplemente consistiría en observar si los requisitos de usuario coinciden con lo que el algoritmo ofrece.

Estas pruebas se encuentran en el punto 4.7.3 Pruebas del algoritmo.

3.1.14 Evaluación del algoritmo (independiente del dominio)

Se procede a comparar los algoritmos elegidos en el punto 4.1.2 Selección de algoritmos con el algoritmo relacionado con la heurística independiente del dominio (probando cada uno de los dominios). Como resulta lógico, es necesario ejecutar previamente dicho algoritmo. Tal y como se ha comentado anteriormente, se determinará que el algoritmo implementado es apropiado cuando sea casi tan bueno o mejor que el algoritmo que proporcione los mejores resultados.

No se ejecutará este algoritmo con otros dominios distintos por 3 principales razones:

1) Los dominios que se iban a estudiar ya se predefinieron.
2) Se supone que, al generalizar usando dominios tan dispares, no resulta del todo necesario utilizar otros dominios distintos para evaluar esta heurística independiente del dominio.
3) Si alguien considerase necesario comparar la heurística con otros dominios distintos, se abriría la posibilidad de realizar otros estudios que pretendan profundizar un poco más en las heurísticas independientes del dominio, continuando con esta misma línea de investigación.

Los algoritmos utilizados para comparar ya se habrán ejecutado anteriormente, por lo que no será necesario realizar ninguna otra ejecución más. Este punto se encuentra íntegro en el proceso de investigación 4.7.4 Evaluación del algoritmo.

3.1.15 Extracción de conclusiones del estudio independiente del dominio

En este paso simplemente se extraerán las conclusiones oportunas tras evaluar los algoritmos con la heurística independiente del dominio implementada. Este punto se desarrolla en la sección 4.7.5 Extracción de conclusiones.

3.1.16 Establecimiento de posibles trabajos futuros

En función de las conclusiones extraídas durante la realización de este proyecto, se podrían anunciar posibles futuras líneas de estudio prometedoras, con el fin de progresar la investigación aquí iniciada. De igual forma, se proponen otras líneas de investigación durante el transcurso del proyecto, al tomar decisiones que encauzan la experimentación. Este punto se desarrolla en la sección Sección 8 Trabajos futuros.

3.2 Alternativas en la metodología de estudio

La metodología de estudio elegida se ha seleccionado en función de ciertos criterios personales. Es posible, por tanto, desarrollar metodologías alternativas tan buenas o mejores que la que se ha llevado a cabo en el proyecto.

En este punto, se nombrarán algunas de ellas agrupadas en dos categorías: modificaciones livianas, y modificaciones drásticas. Estas alternativas también podrían ubicarse dentro de la sección Sección 8 Trabajos futuros, pero se introducen aquí dado que son diferentes formas de abarcar la misma investigación descubiertas durante el desarrollo de la misma metodología.

3.2.1 Modificaciones livianas

En este punto se comentarán posibles modificaciones de la metodología, pero que no suponen un gran cambio, es decir, no variarán demasiado los resultados obtenidos tras la investigación. Por tanto, realizar estas modificaciones en futuras investigaciones no parece, aparentemente, una buena alternativa. A continuación, se listan:

- **Elección de otro sistema operativo**: la elección de otro sistema operativo distinto para la investigación no supone, a primera vista, una decisión que sea determinante para obtener resultados distintos en la investigación.

- **Selección de otro lenguaje de programación**: igual que en la elección de otro sistema operativo, cambiar a otro lenguaje de programación no supondría, a primera vista, una decisión que sea determinante para obtener resultados distintos en la investigación. Esto se debe a que los resultados no radican en la potencia del lenguaje de programación, sino en la comparación entre los distintos resultados.

- **Uso de distintas herramientas auxiliares**: el uso de otros navegadores o editores de texto no supondría ninguna variación en los resultados de la investigación.

- **Elección de otro planificador**: la elección de otro planificador no debería variar los resultados de la investigación, siempre y cuando en este nuevo planificador estén implementados los mismos algoritmos que los empleados en la metodología. Esto se debe a que los resultados no radican en la potencia del planificador, sino en la comparación entre los distintos resultados.

- **Utilizar algoritmos para comparar que no usen PDBs**: dado que el objetivo del proyecto es proponer heurísticas que mejoren los resultados de los algoritmos ya implementados que hacen uso de PDBs, incluir otros algoritmos distintos no cambiará las conclusiones esenciales. Que estos algoritmos ofrezcan resultados mejores o peores a la heurística implementada no proporciona ninguna noción de si es buena idea realizar un estudio del dominio en cuestión usando PDBs. Este proyecto dará respuesta a la pregunta: si el uso de PDBs en cierto dominio es una buena alternativa, ¿será una buena idea hacer un estudio del dominio en cuestión?, y/o ¿sería apropiado hacer uso de ciertas heurísticas independientes del dominio que usen PDBs?

Dicho esto, si en la práctica se observase que el uso de las PDBs no son una de las mejores alternativas, este estudio que profundiza más en ellas no debería proporcionar una respuesta esperanzadora. Sin embargo, cabría pensar en el caso de si sería posible ofrecer algoritmos nuevos sobre PDBs que superen tanto a los actuales; hasta el punto de empezar a usarlos en dominios donde antes su uso en estos era impensable. Podría resultar una prometedora investigación para el futuro, pero se sale de la línea de lo que se espera en este proyecto.

- **Comparaciones entre diferentes lenguajes, planificadores y/o sistemas operativos**: tal y como se ha comentado previamente, no es determinante el uso de otros lenguajes, planificadores o sistemas operativos en el proyecto. Por esa razón, su comparación tampoco tendría que proporcionar unas conclusiones que den respuesta a las preguntas que se desean contestar mediante la realización de esta investigación.

- **Variar la exhaustividad de la experimentación**: proponiendo más problemas, verificando el correcto funcionamiento del algoritmo implementado usando más de 3 problemas por dominio, etc. Sin embargo, no se considera necesario variar la exhaustividad, pues la metodología de la experimentación ya es lo suficientemente pesada y completa.

- **Modificar el esqueleto de la metodología**: a pesar de variar la estructura base de la metodología, los resultados no variarán, por lo que las conclusiones tampoco.

3.2.2 Modificaciones drásticas

En este punto se comentarán posibles modificaciones de la metodología, que suponen un gran cambio. Es decir, podrían variarán los resultados obtenidos tras la investigación. Dicho esto, realizar estas modificaciones en futuras investigaciones podría resultar, aparentemente, una buena alternativa. A continuación, se listan:

- **Usar problemas de satisfabilidad en lugar de problemas optimización**: este pequeño cambio podría suponer una variación de los resultados. Sería interesante abordar esta cuestión en futuras investigaciones que sigan esta línea.

- **Elegir más dominios u otros distintos (de otro sitio, otro año)**: el uso de otros dominios y/o la inclusión de más (de otra colección de dominios distinta a IPC, de otros años, etc.) podría variar los resultados de la investigación. Resultaría interesante proponer otros estudios que intentasen refutar las conclusiones aquí alcanzadas. De esta forma, convergirá a resultados más exactos, todo mediante un estudio empírico. De todos modos, si la investigación desempeñada en este proyecto es acertada, no debería suponer una gran diferencia.

- **Seleccionar nuevos algoritmos de estudio u otros distintos relacionados con PDBs**: cuantos más algoritmos se utilicen para observar las variables relevantes de cada dominio, más sofisticados tenderán a ser los algoritmos que se implementarán. Esto no tiene por qué ser siempre así, pues existen algoritmos que podrían ser despreciados debido a su poca eficacia y/o eficiencia. A pesar de ello, no está de más incluir más algoritmos relacionados con PDBs en futuros estudios.

- **Desarrollar otra heurística independiente del dominio**: futuros proyectos prometedores podrían surgir a raíz de continuar con la investigación aquí iniciada. Estos proyectos podrían enfocarse en obtener otras heurísticas independientes del dominio, mediante la información que aquí se ofrece. Una posible alternativa podría ser desarrollar la heurística independiente del dominio de otra manera que no sea fijarse en la colocación de las variables *relevantes* en la codificación del dominio en cuestión.

- **Evaluar la heurística independiente del dominio con otros dominios**: al generalizar usando dominios tan dispares, no resulta del todo necesario utilizar otros dominios distintos para evaluar esta heurística independiente del dominio. Sin embargo, si alguien considerase necesario comparar la heurística con otros dominios distintos, se abriría la posibilidad de realizar otros estudios que pretendan profundizar un poco más en las heurísticas independientes del dominio, continuando con esta misma línea de investigación. Es difícil afirmar rotundamente si esta decisión es liviana o drástica.

- **Comparaciones entre distintas metodologías**: comparar resultados ayuda a converger a soluciones más exactas. Estas comparaciones podrían realizarse entre problemas de satisfabilidad y optimización, entre algoritmos, dominios, heurísticas independientes del dominio, etc.

- **Desarrollar otros tipos de heurísticas**: en lugar de heurísticas dependientes e independientes del dominio, incluir algún tipo de heurística distinta (siempre y cuando esté relacionada con PDBs) podría proporcionar resultados interesantes.

3.3 Código implementado

3.3.1 Consideraciones generales del código fuente

Antes de nada, es necesario aclarar una cuestión respecto al nombre de los problemas. Este se ha cambiado siguiendo un determinado patrón:

p<número del problema><Identificador del dominio>.pddl

El número del problema tiene dos dígitos y está comprendido desde el 01 hasta el 20 excepto en el *Tetris*, los cuales solo llegan hasta el 17. El orden de designación de estos viene indicado por el orden alfabético de colocación de ficheros en *Ubuntu*. Es decir, después de que *Ubuntu* ordenase los ficheros de forma automática en un directorio, estos se fueron desginando en ese mismo orden. Por otra parte, el identificador del dominio es *S* en el dominio *Snake*, *T* en *Termes*, *H* en *Hiking*, *SP* en *Spider* y *TT* en *Tetris*.

Por otra parte, para la correcta lectura de los dominios y de los problemas cuando se requiera recoger infomación de ahí, es necesario estructurarlos cumpliendo las siguientes premisas:

- Dominios:

  - ⇨ En una sola línea debe declararse la acción correspondiente,
  - ⇨ debajo se introducen los parámetros,
  - ⇨ luego se declaran las precondiciones,

- ⇨ en la línea inferior se introduce la palabra *and*,
- ⇨ en cada línea se introduce una precondición,
- ⇨ antes de declarar los efectos se cierran las precondiciones con un paréntesis,
- ⇨ se declaran los efectos,
- ⇨ se vuelve a introducir la sentencia *and*,
- ⇨ en cada línea se introduce un efecto,
- ⇨ se cierra con un paréntesis,
- ⇨ se cierra la acción con otro paréntesis.
- ⇨ No se introducen espacios adicionales dentro de una acción, los presentes ya son suficientes.
- ⇨ Los *not equal*, *increases* y sentencias *when* (entre otros) van al final de una línea, con el fin de ocultarlos, pues no se recopila ninguna información de ellos. De esta forma, la ordenación podría realizarse en cualquier dominio, no rompiendo la independencia del dominio en el *script* a desarrollar.

- Problemas:

- ⇨ Las metas de los problemas deben tener en una línea la sentencia *goal*,
- ⇨ en la siguiente el *and*,
- ⇨ después cada predicado,
- ⇨ y para finalizar un paréntesis de cierre
- ⇨ De igual forma, no introducir espacios adicionales. Los presentes ya son suficientes. De esta forma, la ordenación podría realizarse en cualquier problema, no rompiendo la independencia del dominio en el *script* a desarrollar.

Por cada uno de los dominios se ha implementado un *script*. En los siguientes puntos se abordarán cuestiones relacionados con estos *scripts*. Se comentarán los requisitos y los casos de uso, la arquitectura del sistema y las funciones implementadas. Respecto a los requisitos y casos de uso se hablará de los requisitos de usuario a alto nivel, así como de los casos de uso y el manual de usuario, de los requisitos funcionales a bajo nivel, de los requisitos no funcionales y, por último, de la matriz de trazabilidad entre requisitos y casos de uso.

Tras esto, se hablará del *script* del algoritmo independiente del dominio, así como de las modificaciones en *iPDB*, siguiendo la misma estructura que antes. Respecto a las modificaciones de *iPDB* tan solo se abordarán cuestiones relacionadas con esa modificación, no con el algoritmo *iPDB* en sí. En último lugar se comentarán algunos aspectos relacionados con los *scripts* utilizados para automatizar las llamadas a los algoritmos implementados y a los de *Fast Downward*, así como de los *scripts* adicionales para agilizar el trabajo.

Cabe destacar que muchos de estos *scripts* no están relacionados con el proyecto en sí, por lo que no tienen una especificación tan detallada. Estos resultan ser algoritmos de apoyo para el entendimiento del código y para la contemplación y recopilación de

resultados. Por otra parte, los algoritmos implementados no son ni complejos ni largos. Acatan la programación funcional, pero no se atienen a la estructuración por clases, por lo que muchos de los puntos anteriormente mencionados no aplicarán.

3.3.2 Plantillas de requisitos y casos de uso

Hay tres tipos de requisitos en el proyecto: de usuario a alto nivel, funcionales a bajo nivel y no funcionales. Todos siguen el mismo formato de tabla. Sin embargo, los casos de uso utilizan una tabla completamente distinta.

3.3.2.1. Plantilla de requitos

Estas plantillas siguen un formato similar al que se siguió en el trabajo de la asignatura de la universidad *Dirección de proyectos de desarrollo de software* (NODDE, 2017).

Los campos de la tabla son:

- **Identificador.** Cada requisito cuenta con un código único que lo identifica. Este sigue el formato *R<id>-<id-dominio><número del requisito>*, donde:

  - ***R*:** referente a requisito.
  - ***<id>***: puede otorgar los valores *U, F* o *NF*, referentes a requisitos de usuario a alto nivel, funcionales a najo nivel o no funcionales, respectivamente.
  - ***<id-dominio>***: puede tomar los valores *S, T, H SP, TT* y *DI*, dependiendo del dominio con el dominio asociado al requisito: *Snake*, *Termes*, *Hiking*, *Spider*, *Tetris* o independiente del dominio respectivamente.
  - ***<número del requisito>*:** es un número que indica el número de requisitos de categoría *<id>*.

- **Nombre:** muestra el nombre del requisito.

- **Descripción.** Define el requisito de forma completa.

- **Prioridad.** Indica el grado de *prioridad* de un requisito con respecto a los demás. Puede ser *alta, media* o *baja.* La prioridad es especialmente importante a la hora de aplicar los requisitos durante el proceso de desarrollo *software*. Se aplican primero aquellos de prioridad *alta*, después *media* y, por último, *baja.*

- **Necesidad.** Determina el grado de importancia de aplicar un requisito. Puede ser:

  - *Esencial.* El requisito debe aplicarse siempre.

- ⇨ *Deseable.* Sería conveniente que el requisito se aplicase, pero no es imprescindible.
- ⇨ *Opcional.* Es irrelevante para el proyecto que el requisito se aplique o no.

- **Claridad.** La falta de claridad implica ambigüedad. El uso de términos específicos, diagramas e ilustraciones pueden ayudar a clarificar el significado del requisito. Puede ser alta, media o baja.

- **Verificabilidad.** Es posible comprobar si el requisito es aplicado por el *software*. Puede ser alta, media o baja.

- **Estabilidad.** Indica si el requisito está abierto o no a posibles cambios.

- **Fuente.** Indica la procedencia del requisito. Puede ser mía, José González Barroso, o del jefe de proyecto Daniel Borrajo Millán.

- **Responsable.** Personal encargado de la creación del requisito.

R<id>-<id-dominio><número del requisito>			
Nombre			
Descripción			
Prioridad	☐ Alta ☐ Media ☐ Baja	**Necesidad**	☐ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☐ Media ☐ Baja	**Verificabilidad**	☐ Alta ☐ Media ☐ Baja
Estabilidad	☐ Si ☐ No	**Fuente**	☐ Interno ☐ Daniel Borrajo
Responsable			

Tabla 3. Plantilla de requisitos

3.3.2.2. Plantilla de casos de uso

Estas plantillas (*tablas 3 y 4*) siguen el mismo formato que el que se siguió en el trabajo de la asignatura de la universidad *Dirección de proyectos de desarrollo de software*. Este se incluye en las referencias.

Los campos de la tabla son:

- **Identificador:** cada caso de uso cuenta con un código único que lo identifica. Este sigue el formato *CU-<id-dominio>XX*, donde:

  - ⇨ ***CU***: referente a caso de uso.
  - ⇨ ***<id-dominio>***: puede tomar los valores *S, T, H SP, TT* y *DI*, dependiendo del dominio con el dominio asociado al requisito: *Snake*, *Termes*, *Hiking*, *Spider*, *Tetris* o independiente del dominio respectivamente.
  - ⇨ ***<número del caso de uso>***: es un número de un dígito, que indica el número de caso de uso en cuestión.

- **Nombre:** nombre del caso de uso en cuestión.

- **Descripción:** define el caso de uso de forma completa.

- **Actor/es**: comenta que Actor o Actores están vinculados con dicho caso de uso.

- **Requisito/s relacionado/s**: muestra de qué requisito se ha extraído el caso de uso.

- **Precondiciones**: muestra las condiciones previas que hay que cumplir para poder realizar el caso de uso.

- **Postcondiciones**: muestra los resultados que lleva a cabo el caso de uso.

CU-<id-dominio><número del caso de uso>	
Nombre	
Descripción	
Actor/es	
Requisito/s relacionados/s	
Precondiciones	
Postcondiciones	

Tabla 4. Plantilla de casos de uso

3.3.3 *Script* de algoritmo para *Snake*

A continuación, se abordará el *script* implementado para el dominio *Snake*, llamado *AlgSnake*.

3.3.3.1. Requisitos y casos de uso

3.3.3.1.1. Requisitos de usuario a alto nivel

El número de requisitos de usuario es solo 2:

RU-S1			
Nombre	Obtener PDBs en *Snake*.		
Descripción	Dado un problema, el *script* deberá ser capaz de proporcionar un conjunto de patrones de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio *Snake*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 5. Requisito de usuario 1 – Snake

RU-S2			
Nombre	Ejecutar planificador con las PDBs obtenidas en *Snake*.		
Descripción	Se ejecutará el planificador de *Fast Downward* (usando *Canonical* y/o *Zero-One*) introduciendo por parámetro las PDBs obtenidas, el dominio *Snake* y el problema el resolver.		
Prioridad	☑ Alta ☐ Media	**Necesidad**	☑ Esencial ☐ Deseable

	☐ Baja		☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 6. Requisito de usuario 2 – Snake

3.3.3.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-S1	
Nombre	Resolver problema en *Snake*.
Descripción	Dado un problema, resolverlo mediante *AlgSnake*.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con Snake.
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el *script AlgSnake*, el dominio Snake y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el *output.sas* y el plan (en caso de encontrar solución)

Tabla 7. Caso de uso – Snake

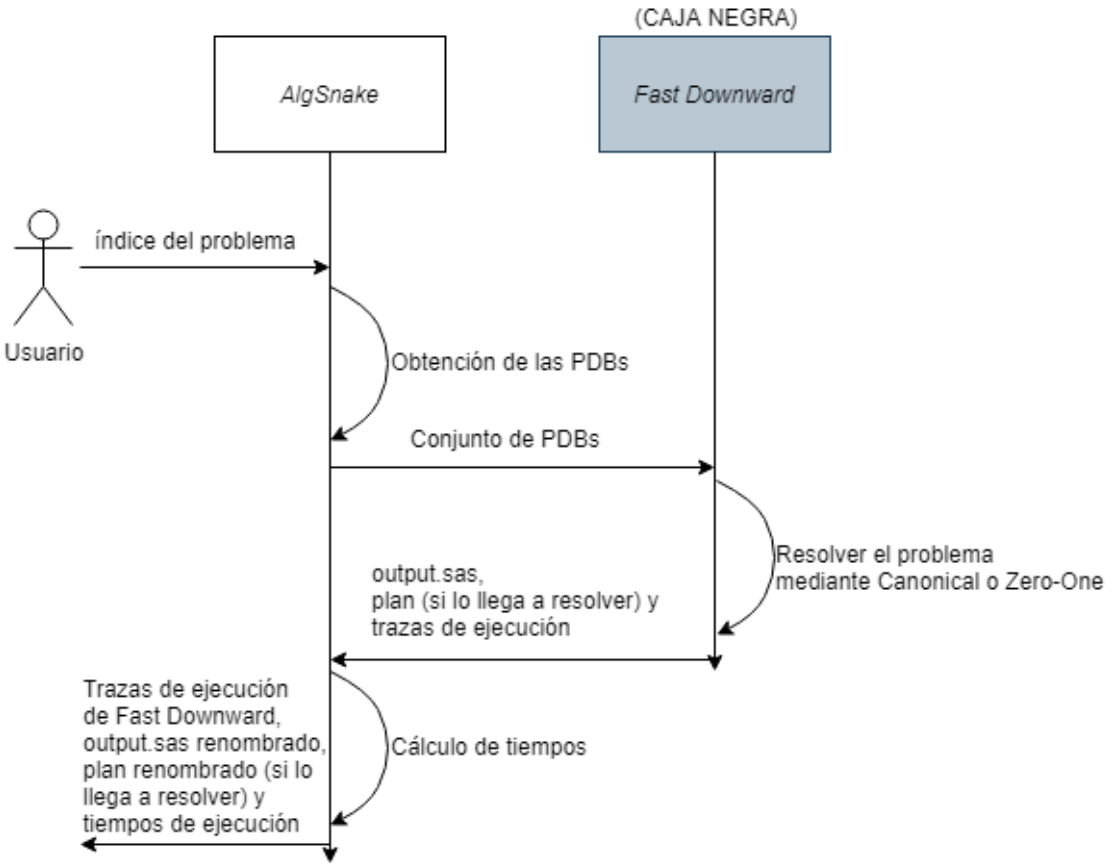

Ilustración 1. Diagrama de casos de uso - Snake

3.3.3.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde se encuentra este, el dominio y el problema a resolver y escribir por consola el siguiente comando:

python AlgSnake.py <número del problema>

Donde *<número del problema>* son dos dígitos que hacen referencia al número del problema a resolver. Por ejemplo: 01, 09, 17.

3.3.3.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF-S01			
Nombre	Recibir el número del problema a resolver – *Snake*.		
Descripción	Recibir e interpretar, al ejecutar *Snake*, el número del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 8. Requisito de software 1 – Snake

RF-S02			
Nombre	Crear *output* – *Snake*.		
Descripción	Crear *output* del problema de *Snake* para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 9. Requisito de software 2 – Snake

RF-S03			
Nombre	Identificar variable *headsnake – Snake*.		
Descripción	Identificar la variable *headsnake* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 10. Requisito de software 3 – Snake

RF-S04			
Nombre	Identificar variable *spawn – Snake*.		
Descripción	Identificar la variable *spawn* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 11. Requisito de software 4 – Snake

RF-S05			
Nombre	Identificar variables con puntos de comida iniciales – *Snake*.		
Descripción	Identificar las variabled con los puntos de comida iniciales en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 12. Requisito de software 5 – Snake

RF-S06			
Nombre	Identificar variables con puntos de comida no iniciales – *Snake*.		
Descripción	Identificar las variables con los puntos de comida no iniciales en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 13. Requisito de software 6 – Snake

RF-S07			
Nombre	Crear PDBs individuales – *Snake*.		
Descripción	Crear una PDB por cada punto de comida inicial.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 14. Requisito de software 7 – Snake

RF-S08			
Nombre	Crear PDBs *largas* – *Snake*.		
Descripción	Crear 9 PDBs con *headsnake*, *spawn*, algunos puntos de comida iniciales y algunos puntos de comida no iniciales, hasta llegar a 14. Se procura que entre todas las PDBs se encuentren todos esos puntos de comida, aunque las PDBs sobrepasen ligeramente las 14 variables.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 15. Requisito de software 8 – Snake

RF-S09			
Nombre	Ordenar PDBs – *Snake*.		
Descripción	Ordenar todas las PDBs de mayor a menor tamaño.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 16. Requisito de software 9 – Snake

RF-S10			
Nombre	Llamar al planificador – *Snake*.		
Descripción	Ejecutar el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs obtenidas.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 17. Requisito de software 10 – Snake

RF-S11			
Nombre	Guardar *output* – *Snake*.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 18. Requisito de software 11 – Snake

RF-S12			
Nombre	Guardar plan solución – *Snake*.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 19. Requisito de software 12 – Snake

RF-S13			
Nombre	Obtener tiempos – *Snake*.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de construcción de las PDB, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 20. Requisito de software 13 – Snake

3.3.3.1.5. Requisitos no funcionales

A continuación, se muestran los requisitos no funcionales del sistema:

RNF-S01			
Nombre	Aceptación de *AlgSnake*.		
Descripción	Aceptar *AlgSnake* como algoritmo válido siempre y cuando supere o alcance la calidad del mejor algoritmo del proceso de investigación inicial sobre el dominio *Snake*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 21. Requisito no funcional 1 – Snake

RNF-S02			
Nombre	Plan de pruebas - *AlgSnake*.		
Descripción	Elegir 3 problemas distintos y observar la salida esperada y obtenida por *AlgSnake*, respecto a las PDBs que se obtienen. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 22. Requisito no funcional 2 – Snake

3.3.3.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada se comprondría tan solo de una fila.

3.3.3.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12	13
1	X	X	X	X	X	X	X	X	X				X
2	X									X	X	X	X

Tabla 23. Matriz de trazabilidad – Snake

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.3.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.3.1.2 Casos de uso y en la descripción de funciones implementadas 3.3.3.3 Funciones implementadas se puede entender el código del *script* implementado.

3.3.3.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(numeroProblema)***: crea el archivo *output.sas* del problema.

- ***obtencionPDBs(numeroProblema)***: esta función crea las PDBs necesarias para intentar resolver el problema en cuestión del dominio *Snake*. A continuación, se muestra el procedimiento de actuación del algoritmo en ese proceso de creación de PDBs:

- ⇨ Abrir el problema e identificar los puntos de comida iniciales, es decir, los que se pueden devorar al inicio.
- ⇨ Abrir el *output*, leerlo e identificar la variable *headsnake*, la variable *spawn* y los puntos de comida iniciales y no iniciales.
- ⇨ Crear una PDB por cada punto de comida inicial.
- ⇨ Crear unas 9 PDBs con *headsnake*, *spawn*, algunos puntos de comida iniciales y algunos puntos de comida no iniciales, hasta llegar a 14. Se procura que entre todas las PDBs se encuentren todos esos puntos de comida, aunque las PDBs sobrepasen ligeramente las 14 variables.
- ⇨ Ordenar PDBs de mayor a menor tamaño.
- ⇨ Devolver el conjunto de PDBs.

- **ejecucionPlanificador(pdbs,numeroProblema)**: ejecuta el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs.

- **guardarArchivos(numeroProblema)**: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones.

- **main()**: ejecuta las funciones *crearOutput*, *obtencionPDBs*, *ejecucionPlanificador* y *guardarArchivos* en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.4 *Script* de algoritmo para *Termes*

A continuación, se abordarán el *script* impementado para el dominio *Termes*, llamado *AlgTermes*.

3.3.4.1. Requisitos y casos de uso

3.3.4.1.1. Requisitos de usuario a alto nivel

El número de requisitos de usuario es solo 2:

RU-T1			
Nombre	Obtener PDBs en *Termes*.		
Descripción	Dado un problema, el *script* deberá ser capaz de proporcionar un conjunto de patrones de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio *Termes*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 24. Requisito de usuario 1 – Termes

RU-T2			
Nombre	Ejecutar planificador con las PDBs obtenidas en *Termes*.		
Descripción	Se ejecutará el planificador de Fast Downward (usando *Canonical* y/o *Zero-One*) introduciendo por parámetro las PDBs obtenidas, el dominio Termes y el problema el resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 25. Requisito de usuario 2 – Termes

3.3.4.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-T1	
Nombre	Resolver problema en *Termes*.
Descripción	Dado un problema, resolverlo mediante *AlgTermes*.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con *Termes*.
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el *script AlgTermes*, el dominio *Termes* y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el *output.sas* y el plan (en caso de encontrar solución)

Tabla 26. Caso de uso – Termes

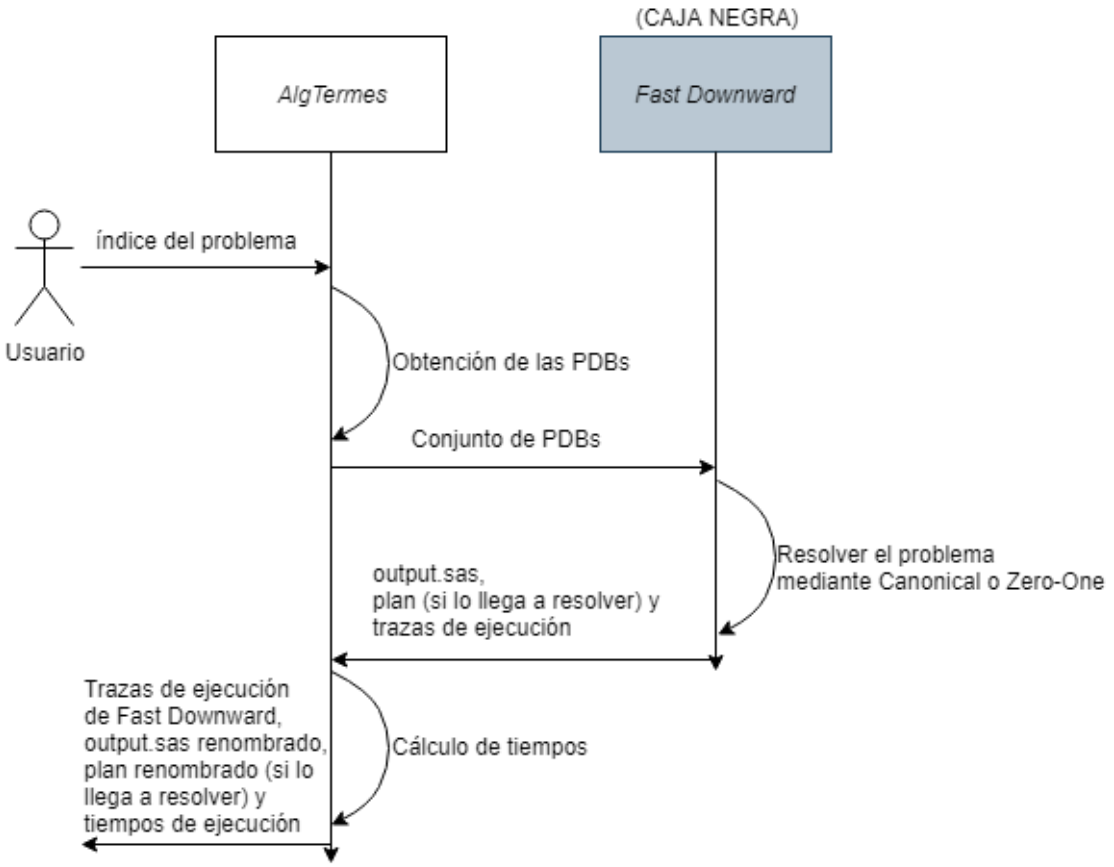

Ilustración 2. Diagrama de casos de uso - Termes

3.3.4.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde este se encuentra, el dominio y el problema a resolver y escribir por consola el siguiente comando:

python AlgTermes.py <número del problema>

Donde <número del problema> son dos dígitos que hacen referencia al número del problema a resolver. Por ejemplo: 01, 09, 17.

3.3.4.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF-T01			
Nombre	Recibir el número del problema a resolver – *Termes*.		
Descripción	Recibir e interpretar, al ejecutar *Termes*, el número del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 27. Requisito de software 1 – Termes

RF-T02			
Nombre	Crear *output* – *Termes*.		
Descripción	Crear *output* del problema de *Termes* para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 28. Requisito de software 2 – Termes

RF-T03			
Nombre	Identificar variable con la posición del robot – *Termes*.		
Descripción	Identificar la variable con la posición del robot en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 29. Requisito de software 3 – Termes

RF-T04			
Nombre	Identificar variable *has-block* – *Termes*.		
Descripción	Identificar la variable *has-block* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 30. Requisito de software 4 – Termes

RF-T05			
Nombre	Identificar variables con las *alturas variables – Termes*.		
Descripción	Identificar las variables con las *alturas variables* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 31. Requisito de software 5 – Termes

RF-T06			
Nombre	Identificar variables con las *alturas no variables – Termes*.		
Descripción	Identificar las variables con las *alturas no variables* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 32. Requisito de software 6 – Termes

RF-T07			
Nombre	Identificar vecinos – *Termes*.		
Descripción	Identificar los vecinos de las posiciones con las *alturas variables*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 33. Requisito de software 7 – Termes

RF-T08			
Nombre	Crear PDBs individuales – *Termes*.		
Descripción	Crear una PDB por cada *altura no variable*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 34. Requisito de software 8 – Termes

RF-T09			
Nombre	Crear PDB *mediana – Termes*.		
Descripción	Crear una PDB con la posición del robot y las *alturas variables*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 35. Requisito de software 9 – Termes

RF-T10			
Nombre	Crear PDBs *largas* – Termes.		
Descripción	Crear una PDB por cada combinación entre los vecinos de las *alturas variables*, introduciendo, además, la posición del robot, el *has-block* y todas las *alturas variables*, siempre y cuando el tamaño no sea superior a 10 variables totales para que el grafo asociado quepa en memoria.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 36. Requisito de software 10 – Termes

RF-T11			
Nombre	Crear PDB *muy larga – Termes*.		
Descripción	Crear una PDB con la posición del robot, el *has-block* y todas las *alturas variables* junto a sus vecinos, siempre y cuando el tamaño no sea superior a 10 variables totales, para que el grafo asociado quepa en memoria.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 37. Requisito de software 11 – Termes

RF-T12			
Nombre	Ordenar PDBs – *Termes*.		
Descripción	Ordenar todas las PDBs de mayor a menor tamaño.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 38. Requisito de software 12 – Termes

RF-T13			
Nombre	Llamar al planificador – *Termes*.		
Descripción	Ejecutar el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs obtenidas.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 39. Requisito de software 13 – Termes

RF-T14			
Nombre	Guardar *output* – *Termes*.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 40. Requisito de software 14 – Termes

RF-T15			
Nombre	Guardar plan solución – *Termes*.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 41. Requisito de software 15 – Termes

RF-T16			
Nombre	Obtener tiempos – *Termes*.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de construcción de las PDB, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 42. Requisito de software 16 – Termes

3.3.4.1.5. Requisitos no funcionales

RNF-T01			
Nombre	Aceptación de *AlgTermes*.		
Descripción	Aceptar *AlgTermes* como algoritmo válido siempre y cuando supere o alcance la calidad del mejor algoritmo del proceso de investigación inicial sobre el dominio *Termes*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 43. Requisito no funcional 1 – Termes

RNF-T02			
Nombre	Plan de pruebas - *AlgTermes*.		
Descripción	Elegir 3 problemas distintos y observar la salida esperada y obtenida por *AlgTermes*, respecto a las PDBs que se obtienen. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 44. Requisito no funcional 2 – Termes

3.3.4.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada tan solo se compondría de una fila.

3.3.4.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16
1	X	X	X	X	X	X	X	X	X	X	X	X				X
2	X												X	X	X	X

Tabla 45. Matriz de trazabilidad – Snake

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.4.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.4.1.2**¡Error! No se encuentra el origen de la referencia.** Casos de uso y en la descripción de funciones implementadas 3.3.4.3**¡Error! No se encuentra el origen de la referencia.** Funciones implementadas se puede entender el código del *script* implementado.

3.3.4.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(numeroProblema)***: crea el archivo *output.sas* del problema.

- ***obtencionPDBs(numeroProblema)***: esta función crea las PDBs necesarias para intentar resolver el problema en cuestión del dominio *Termes*. A continuación, se muestra el procedimiento de actuación del algoritmo en ese proceso de creación de PDBs:

- ⇨ Abrir el problema e identificar el tamaño del tablero, la localización de la cantera (*depot*), y las posiciones con altura variable, es decir, aquellas alturas que tienen un valor distinto en la meta que en el estado inicial.
- ⇨ Abrir el *output*, leerlo e identificar la variable con la posición del robot, el *has-block* y las variables con las alturas.
- ⇨ Crear una PDB por cada *altura no variable*, y una PDB con la posición del robot y las *alturas variables*.
- ⇨ Crear una tabla con todas las *alturas variables* y sus vecinos.
- ⇨ Crear una PDB con la posición del robot, el *has-block* y todas las *alturas variables* junto a sus vecinos, siempre y cuando el tamaño no sea superior a 10 variables totales, para que el grafo asociado quepa en memoria.
- ⇨ Una PDB por cada combinación entre los vecinos de las *alturas variables*, introduciendo, además, la posición del robot, el *has-block* y todas las *alturas variables*, siempre y cuando el tamaño no sea superior a 10 variables totales.
- ⇨ Ordenar PDBs de mayor a menor tamaño.
- ⇨ Devolver el conjunto de PDBs.

- ***ejecucionPlanificador(pdbs,numeroProblema)***: ejecuta el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs.

- ***guardarArchivos(numeroProblema)***: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones.

- ***main()***: ejecuta las funciones *crearOutput*, *obtencionPDBs*, *ejecucionPlanificador* y *guardarArchivos* en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.5 *Script* de algoritmo para *Hiking*

A continuación, se abordarán el *script* impementado para el dominio *Hiking*, llamado *AlgHiking*.

3.3.5.1. Requisitos y casos de uso

3.3.5.1.1. Requisitos de usuario a alto nivel

El número de requisitos de usuario es solo 2:

RU-H1			
Nombre	Obtener PDBs en *Hiking*.		
Descripción	Dado un problema, el *script* deberá ser capaz de proporcionar un conjunto de patrones de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio *Hiking*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 46. Requisito de usuario 1 – Hiking

RU-H2			
Nombre	Ejecutar planificador con las PDBs obtenidas en *Hiking*.		
Descripción	Se ejecutará el planificador de *Fast Downward* (usando *Canonical* y/o *Zero-One*) introduciendo por parámetro las PDBs obtenidas, el dominio *Hiking* y el problema el resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 47. Requisito de usuario 2 – Hiking

3.3.5.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-T1	
Nombre	Resolver problema en *Termes*.
Descripción	Dado un problema, resolverlo mediante *AlgTermes*.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con *Termes*.
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el *script AlgTermes*, el dominio *Termes* y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el *output.sas* y el *plan* (en caso de encontrar solución)

Tabla 48. Caso de uso – Hiking

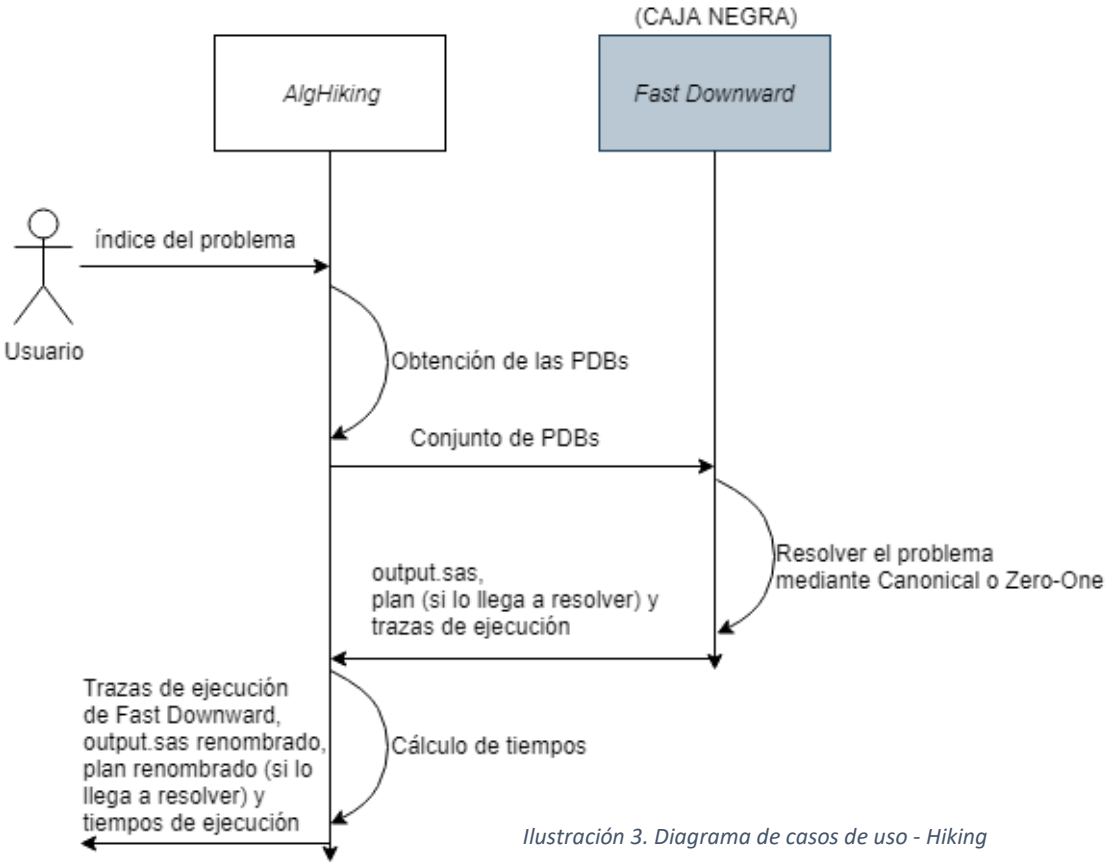

Ilustración 3. Diagrama de casos de uso - Hiking

3.3.5.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde se encuentra, junto al dominio y el problema a resolver, y escribir por consola el siguiente comando:

python AlgHiking.py <número del problema>

Donde <número del problema> son dos dígitos que hacen referencia al número del problema a resolver. Por ejemplo: 01, 09, 17.

3.3.5.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF-H01			
Nombre	Recibir el número del problema a resolver – *Hiking*.		
Descripción	Recibir e interpretar, al ejecutar *Hiking*, el número del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 49. Requisito de software 1 – Hiking

RF-H02			
Nombre	Crear *output* – *Hiking*.		
Descripción	Crear *output* del problema de *Hiking* para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 50. Requisito de software 2 – Hiking

RF-H03			
Nombre	Identificar variables acerca de las tiendas de campaña – *Hiking*.		
Descripción	Identificar las variables acerca de las tiendas de campaña en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 51. Requisito de software 3 – Hiking

RF-H04			
Nombre	Identificar variables *walked – Hiking*.		
Descripción	Identificar las variables *walked* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 52. Requisito de software 4 – Hiking

RF-H05			
Nombre	Identificar variables con la ubicación de cada persona asociado a cada *walked – Hiking*.		
Descripción	Identificar las variables con la ubicación de cada persona asociado a cada *walked* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 53. Requisito de software 5 – Hiking

RF-H06			
Nombre	Crear PDBs individuales – *Hiking*.		
Descripción	Crear una PDB por cada *walked*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 54. Requisito de software 6 – Hiking

RF-H07			
Nombre	Crear PDBs *medianas* – *Hiking*.		
Descripción	Crear una PDB con cada *walked*, que contenga la ubicación de las personas asociadas a dichos *walked* y todas las variables acerca de las tiendas de campaña.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 55. Requisito de software 7 – Hiking

RF-H08			
Nombre	Ordenar PDBs – *Hiking*.		
Descripción	Ordenar todas las PDBs de mayor a menor tamaño.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 56. Requisito de software 8 – Hiking

RF-H09			
Nombre	Llamar al planificador – *Hiking*.		
Descripción	Ejecutar el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs obtenidas.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 57. Requisito de software 9 – Hiking

RF-H10			
Nombre	Guardar *output* – *Hiking*.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 58. Requisito de software 10 – Hiking

RF-H11			
Nombre	Guardar plan solución – *Hiking*.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 59. Requisito de software 11 – Hiking

RF-H12			
Nombre	Obtener tiempos – *Hiking*.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de construcción de las PDB, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 60. Requisito de software 12 – Hiking

3.3.5.1.5. Requisitos no funcionales

RNF-H01			
Nombre	Aceptación de *AlgHiking*.		
Descripción	Aceptar *AlgHiking* como algoritmo válido siempre y cuando supere o alcance la calidad del mejor algoritmo del proceso de investigación inicial sobre el dominio *Hiking*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 61. Requisito no funcional 1 – Hiking

RNF-H02			
Nombre	Plan de pruebas - *AlgHiking*.		
Descripción	Elegir 3 problemas distintos y observar la salida esperada y obtenida por *AlgHiking*, respecto a las PDBs que se obtienen. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 62. Requisito no funcional 2 – Hiking

3.3.5.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada tan solo se compondría de una fila.

3.3.5.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12
1	X	X	X	X	X	X	X	X				X
2	X								X	X	X	X

Tabla 63. Matriz de trazabilidad – Hiking

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.5.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.5.1.2 Casos de uso y en la descripción de funciones implementadas 3.3.5.3 Funciones implementadas se puede entender el código del *script* implementado.

3.3.5.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(numeroProblema)***: crea el archivo *output.sas* del problema.

- ***obtencionPDBs(numeroProblema)***: esta función crea las PDBs necesarias para intentar resolver el problema en cuestión del dominio *Hiking*. A continuación, se muestra el procedimiento de actuación del algoritmo en ese proceso de creación de PDBs:

  - ⇨ Abrir el *output*, leerlo e identificar las variables sobre las tiendas de campaña, los *walked* y la ubicación de las personas asociadas a dichos *walked*.
  - ⇨ Crear una PDB por cada *walked*.
  - ⇨ Crear una PDB con cada *walked*, que contenga la ubicación de las personas asociadas a dichos *walked* y todas las variables acerca de las tiendas de campaña.
  - ⇨ Ordenar PDBs de mayor a menor tamaño.
  - ⇨ Devolver el conjunto de PDBs.

- ***ejecucionPlanificador(pdbs,numeroProblema)*:** ejecuta el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs.

- ***guardarArchivos(numeroProblema)***: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones.

- ***main()***: ejecuta las funciones *crearOutput*, *obtencionPDBs*, *ejecucionPlanificador* y *guardarArchivos* en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.6 *Script* de algoritmo para *Spider*

A continuación, se abordarán el *script* impementado para el dominio *Spider*, llamado *AlgSpider*.

3.3.6.1. Requisitos y casos de uso

3.3.6.1.1. Requisitos de usuario a alto nivel

El número de requisitos de usuario es solo 2:

RU-SP1			
Nombre	Obtener PDBs en *Spider*.		
Descripción	Dado un problema, el *script* deberá ser capaz de proporcionar un conjunto de patrones de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio *Spider*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 64. Requisito de usuario 1 – Spider

RU-SP2			
Nombre	Ejecutar planificador con las PDBs obtenidas en *Spider*.		
Descripción	Se ejecutará el planificador de *Fast Downward* (usando *Canonical* y/o *Zero-One*) introduciendo por parámetro las PDBs obtenidas, el dominio *Spider* y el problema el resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 65. Requisito de usuario 2 – Spider

3.3.6.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-SP1	
Nombre	Resolver problema en *Spider*.
Descripción	Dado un problema, resolverlo mediante *AlgSpider*.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con *Spider*.
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el script *AlgSpider*, el dominio *Spider* y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el *output.sas* y el plan (en caso de encontrar solución)

Tabla 66. Caso de uso – Spider

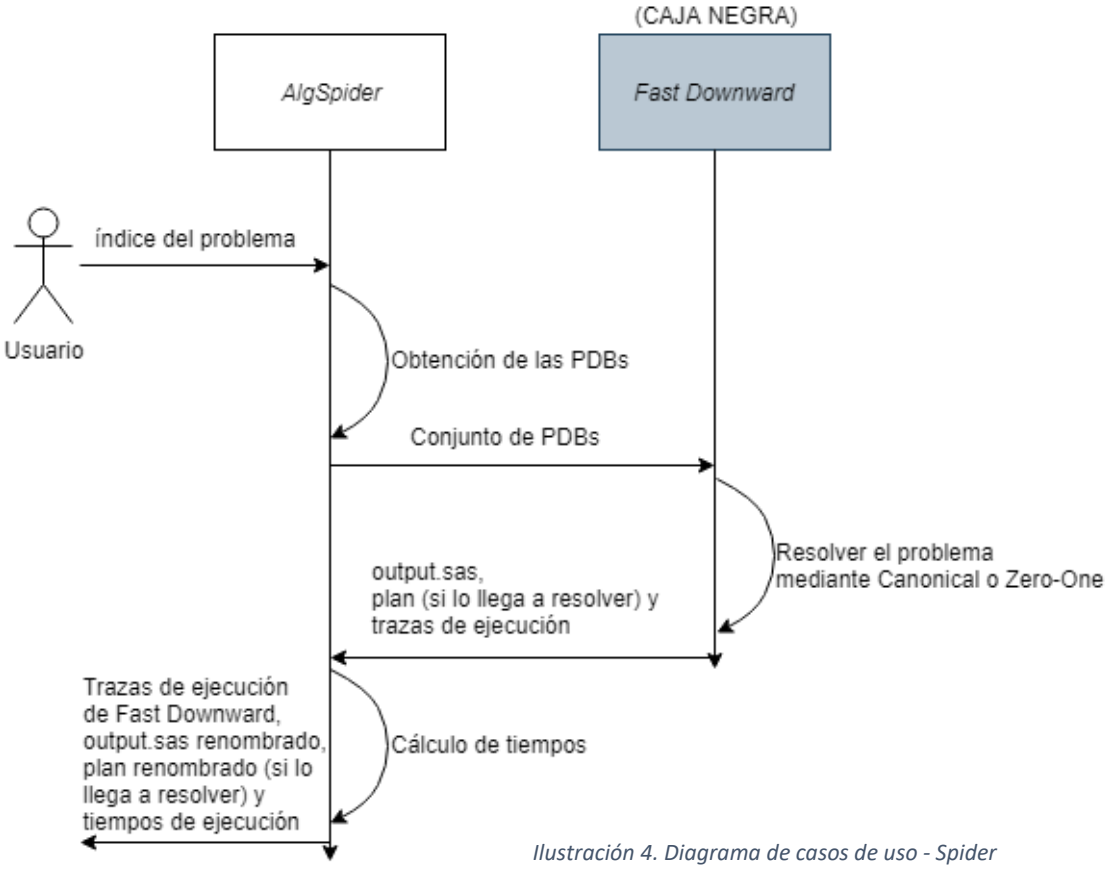

Ilustración 4. Diagrama de casos de uso - Spider

3.3.6.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde se encuentra, junto al dominio y el problema a resolver y escribir por consola el siguiente comando:

python AlgSpider.py <número del problema>

Donde <número del problema> son dos dígitos que hacen referencia al número del problema a resolver. Por ejemplo: 01, 09, 17.

3.3.6.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF- SP01			
Nombre	Recibir el número del problema a resolver – *Spider*.		
Descripción	Recibir e interpretar, al ejecutar *Spider*, el número del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 67. Requisito de software 1 – Spider

RF- SP02			
Nombre	Crear *output* – *Spider*.		
Descripción	Crear *output* del problema de *Spider* para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 68. Requisito de software 2 – Spider

RF- SP03			
Nombre	Identificar variables acerca de las cartas de *pile* – *Spider*.		
Descripción	Identificar la variable acerca de las cartas de *pile* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 69. Requisito de software 3 – Spider

RF- SP04			
Nombre	Identificar variables acerca de las cartas de *deal – Spider*.		
Descripción	Identificar la variable acerca de las cartas de *deal* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 70. Requisito de software 4 – Spider

RF- SP05			
Nombre	Identificar variables acerca de las *clear* de *pile – Spider*.		
Descripción	Identificar la variable acerca de los *clear* de *pile* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 71. Requisito de software 5 – Spider

RF- SP06			
Nombre	Identificar variables acerca de los *clear* de *deal – Spider*.		
Descripción	Identificar las variables acerca de los *clear* de *deal* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 72. Requisito de software 6 – Spider

RF- SP07			
Nombre	Identificar variable *currently-collecting-deck – Spider*.		
Descripción	Identificar la variable *currently-collecting-deck* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 73. Requisito de software 7 – Spider

RF- SP08			
Nombre	Identificar variable *currently-dealing – Spider*.		
Descripción	Identificar la variable *currently-dealing* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 74. Requisito de software 8 – Spider

RF- SP09			
Nombre	Identificar variable *current-deal – Spider*.		
Descripción	Identificar la variable variable *current-deal* en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 75. Requisito de software 9 – Spider

RF- SP10			
Nombre	Crear PDBs individuales – *Spider*.		
Descripción	Crear una PDB individual por cada variable de cartas de *deal* y de *pile* y por cada *clear* de *deal* y de *pile*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 76. Requisito de software 10 – Spider

RF- SP11			
Nombre	Crear PDBs *pequeñas* – *Spider*.		
Descripción	Crear una PDB por cada carta de *pile*, combinado con el *currently-collecting-deck* y dicha carta.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 77. Requisito de software 11 – Spider

RF- SP12			
Nombre	Crear PDBs *medianas – Spider*.		
Descripción	Crear una PDB por cada *clear* de *deal*, combinado con el *current-deal*, el *currently-dealing* y dicho *clear*. También crear una PDB por cada carta de *deal*, combinado con el *current-deal*, el *currently-dealing* y dicha carta.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 78. Requisito de software 12 – Spider

RF- SP13			
Nombre	Crear PDBs medianas *más largas – Spider*.		
Descripción	Crear una PDB por cada combinación entre *clear* de *deal* y carta de *deal*, introduciendo también el *current-deal* y el *currently-dealing*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 79. Requisito de software 13 – Spider

RF- SP14			
Nombre	Ordenar PDBs – *Spider*.		
Descripción	Ordenar todas las PDBs de mayor a menor tamaño.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 80. Requisito de software 14 – Spider

RF- SP15			
Nombre	Llamar al planificador – *Spider*.		
Descripción	Ejecutar el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs obtenidas.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 81. Requisito de software 15 – Spider

RF- SP16			
Nombre	Guardar *output* – *Spider*.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 82. Requisito de software 16 – Spider

RF- SP17			
Nombre	Guardar plan solución – *Spider*.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 83. Requisito de software 17– Spider

RF-SP18			
Nombre	Obtener tiempos – *Spider*.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de construcción de las PDB, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 84. Requisito de software 18 – Spider

3.3.6.1.5. Requisitos no funcionales

RNF-SP01			
Nombre	Aceptación de *AlgSpider*.		
Descripción	Aceptar *AlgSpider* como algoritmo válido siempre y cuando supere o alcance la calidad del mejor algoritmo del proceso de investigación inicial sobre el dominio *Spider*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 85. Requisito no funcional 1 – Spider

RNF-SP02			
Nombre	Plan de pruebas - *AlgSpider*.		
Descripción	Elegir 3 problemas distintos y observar la salida esperada y la obtenida por *AlgSpider*, respecto a las PDBs que se generan. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 86. Requisito no funcional 2 – Spider

3.3.6.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y este se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada se compondría solo de una fila.

3.3.6.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15	16	17	18
1	X	X	X	X	X	X	X	X	X	X	X	X	X	X				X
2	X														X	X	X	X

Tabla 87. Matriz de trazabilidad – Spider

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.6.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.6.1.2 Casos de uso y en la descripción de funciones implementadas 3.3.6.3 Funciones implementadas se puede entender el código del *script* implementado.

3.3.6.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(numeroProblema)***: crea el archivo *output.sas* del problema.

- ***obtencionPDBs(numeroProblema)***: esta función crea las PDBs necesarias para intentar resolver el problema en cuestión del dominio *Spider*. A continuación, se muestra el procedimiento de actuación del algoritmo en ese proceso de creación de PDBs:

- ⇨ Abrir el problema, leerlo e identificar las cartas de *pile*.
- ⇨ Abrir el *output*, leerlo e identificar las variables con las cartas de *deal*, las cartas de *pile*, los clear de *deal*, los clear de *pile*, el *currently-collecting-deck*, *current-deal* y el *currently-dealing*.
- ⇨ Crear una PDB individual por cada variable de cartas de *deal* y de *pile* y por cada *clear* de *deal* y de *pile*.
- ⇨ Crear una PDB por cada carta de *pile*, combinado con el cur*rently-collecting-deck* y dicha carta.
- ⇨ Crear una PDB por cada carta de *deal*, combinado con el *current-deal*, el *currently-dealing* y dicha carta.
- ⇨ Crear una PDB por cada clear de *deal*, combinado con el *current-deal*, el *currently-dealing* y dicho *clear*.
- ⇨ Crear una PDB por cada combinación entre *clear* de *deal* y carta de *deal*, introduciendo también el *current-deal* y el *currently-dealing*.
- ⇨ Ordenar PDBs de mayor a menor tamaño.
- ⇨ Devolver el conjunto de PDBs.

- ***ejecucionPlanificador(pdbs,numeroProblema)***: ejecuta el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs.

- ***guardarArchivos(numeroProblema)***: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones.

- ***main()***: ejecuta las funciones *crearOutput*, *obtencionPDBs*, *ejecucionPlanificador* y *guardarArchivos* en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.7 *Script* de algoritmo para *Tetris*

A continuación, se abordarán el *script* impementado para el dominio *Tetris*, llamado *AlgTetris*.

3.3.7.1. Requisitos y casos de uso

3.3.7.1.1. Requisitos de usuario a alto nivel

El número de requisitos de usuario es solo 2:

RU-TT1			
Nombre	Obtener PDBs en *Tetris*.		
Descripción	Dado un problema, el *script* deberá ser capaz de proporcionar un conjunto de patrones de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio *Tetris*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 88. Requisito de usuario 1 – Tetris

RU-TT2			
Nombre	Ejecutar planificador con las PDBs obtenidas en *Tetris*.		
Descripción	Se ejecutará el planificador de *Fast Downward* (usando *Canonical* y/o *Zero-One*) introduciendo por parámetro las PDBs obtenidas, el dominio *Tetris* y el problema el resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 89. Requisito de usuario 2 – Tetris

3.3.7.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-TT1	
Nombre	Resolver problema en *Tetris*.
Descripción	Dado un problema, resolverlo mediante *AlgTetris*.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con *Tetris*.
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el *script AlgTetris*, el dominio *Tetris* y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el *output.sas* y el plan (en caso de encontrar solución).

Tabla 90. Caso de uso – Tetris

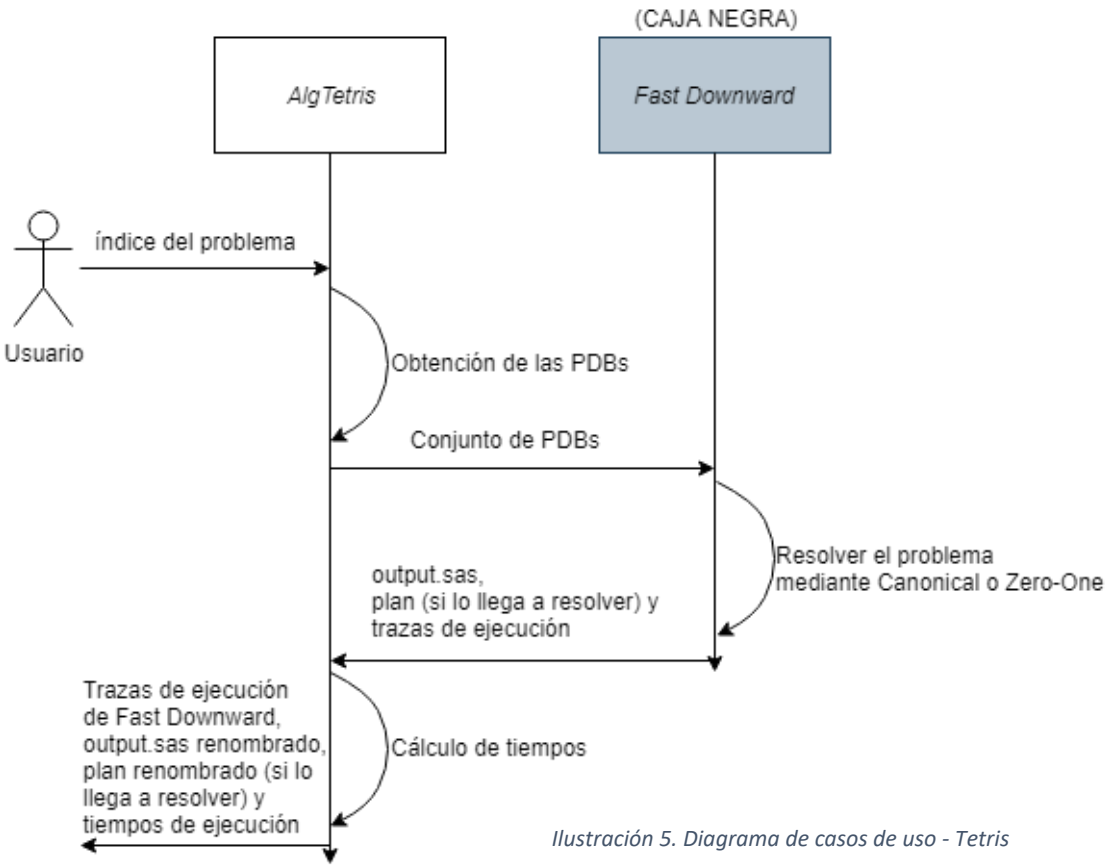

Ilustración 5. Diagrama de casos de uso - Tetris

3.3.7.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde se encuentra, junto al dominio y el problema a resolver y escribir por consola el siguiente comando:

python AlgTetris.py <número del problema>

Donde <número del problema> son dos dígitos que hacen referencia al número del problema a resolver. Por ejemplo: 01, 09, 17.

3.3.7.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF- TT01			
Nombre	Recibir el número del problema a resolver – *Tetris*.		
Descripción	Recibir e interpretar, al ejecutar *Tetris*, el número del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 91. Requisito de software 1 – Tetris

RF- TT02			
Nombre	Crear *output* – *Tetris*.		
Descripción	Crear *output* del problema de *Tetris* para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 92. Requisito de software 2 – Tetris

RF- TT03			
Nombre	Identificar variables *clear* meta – *Tetris*.		
Descripción	Identificar las variables *clear* meta en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 93. Requisito de software 3 – Tetris

RF- TT04			
Nombre	Identificar variables *clear* no meta – *Tetris*.		
Descripción	Identificar las variables *clear* no meta en el fichero *output* creado.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 94. Requisito de software 4 – Tetris

RF- TT05			
Nombre	Crear el tablero con sus variables – *Tetris*.		
Descripción	Crear un tablero con las mismas dimensiones que el problema a resolver, introduciendo en cada posición el número de la variable *clear* asociada a dicha posición.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 95. Requisito de software 5 – Tetris

RF- TT06			
Nombre	Crear PDBs *medianas – Tetris*.		
Descripción	Crear una PDB por columna del tablero, introduciendo todas las variables de dicha columna.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 96. Requisito de software 6 – Tetris

RF- TT07			
Nombre	Crear PDBs *largas – Tetris*.		
Descripción	Crear una PDB que contenga 19 variables, incluyendo todas metas desde la esquina superior izquierda del tablero. En caso de no incluir todas las metas en esa PDB, crear otra PDB de 19 veriables siguiendo el mismo orden, de izquierda a derecha, hasta que se introduzcan todas las metas o hasta que se introduzcan todas las variables.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 97. Requisito de software 7 – Tetris

RF- TT08			
Nombre	Ordenar PDBs – *Tetris*.		
Descripción	Ordenar todas las PDBs de mayor a menor tamaño.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 98. Requisito de software 8 – Tetris

RF- TT09			
Nombre	Llamar al planificador – *Tetris*.		
Descripción	Ejecutar el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs obtenidas.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 99. Requisito de software 9 – Tetris

RF- TT10			
Nombre	Guardar *output* – *Tetris*.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 100. Requisito de software 10 – Tetris

RF- TT11			
Nombre	Guardar plan solución – *Tetris*.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 101. Requisito de software 11 – Tetris

RF-TT12			
Nombre	Obtener tiempos – *Tetris*.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de construcción de las PDB, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 102. Requisito de software 12 – Tetris

3.3.7.1.5. Requisitos no funcionales

RNF-TT01			
Nombre	Aceptación de *AlgTetris*.		
Descripción	Aceptar *AlgTetris* como algoritmo válido siempre y cuando supere o alcance la calidad del mejor algoritmo del proceso de investigación inicial sobre el dominio *Tetris*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 103. Requisito no funcional 1 – Termes

RNF-TT02			
Nombre	Plan de pruebas - *AlgTetris*.		
Descripción	Elegir 3 problemas distintos y observar la salida esperada y obtenida por *AlgTetris*, respecto a las PDBs que se generan. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 104. Requisito no funcional 2 – Termes

3.3.7.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y este se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada se compondría de una sola fila.

3.3.7.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12
1	X	X	X	X	X	X	X	X				X
2	X								X	X	X	X

Tabla 105. Matriz de trazabilidad – Tetris

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.7.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.7.1.2 Casos de uso y en la descripción de funciones implementadas 3.3.7.3 Funciones implementadas se puede entender el código del *script* implementado.

3.3.7.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(numeroProblema)***: crea el archivo *output*.*sas* del problema.

- ***obtencionPDBs(numeroProblema)***: esta función crea las PDBs necesarias para intentar resolver el problema en cuestión del dominio *Tetris*. A continuación, se muestra el procedimiento de actuación del algoritmo en ese proceso de creación de PDBs:

  - ⇨ Abrir el problema, leerlo, identificar el tamaño del tablero y los *clear* que aparecen en la meta.
  - ⇨ Crear una matriz que simule el tablero del *Tetris*.
  - ⇨ Abrir el *output*, leerlo e identificar las variables *clear*, introduciéndolas en la matriz creada.
  - ⇨ Crear una PDB que contenga 19 variables, incluyendo todas metas desde la esquina superior izquierda del tablero. En caso de no incluir todas las metas en esa PDB, crear otra PDB de 19 variables siguiendo el mismo orden, de izquierda a derecha, hasta que se introduzcan todas las metas o hasta que se introduzcan todas las variables.
  - ⇨ Crear una PDB por columna del tablero, introduciendo todas las variables de dicha columna.
  - ⇨ Ordenar PDBs de mayor a menor tamaño.
  - ⇨ Devolver el conjunto de PDBs.

- ***ejecucionPlanificador(pdbs,numeroProblema)***: ejecuta el algoritmo *Canonical* o el *Zero-One* introduciendo directamente las PDBs.

- ***guardarArchivos(numeroProblema)***: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones.

- ***main()***: ejecuta las funciones *crearOutput*, *obtencionPDBs*, ejecucion*P*lanificador y *guardarArchivos* en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.8 *Script* del algoritmo independiente del dominio

A continuación, se abordará el *script* implementado para reunir las variables relevantes y dárselas a *iPDB* modificado, que se trata del mismo algoritmo *iPDB* pero con ciertas modificaciones realizadas en la investigación del punto 4.7.1 Observación de características similares entre las PDBs relevantes de cada dominio.

3.3.8.1. Requisitos y casos de uso

3.3.8.1.1. Requisitos de usuario a alto nivel

Debido a la simplez del Algoritmo y a su propósito, el número de requisitos de usuario es solo 2:

RU-DI1			
Nombre	Obtener *variables relevantes*.		
Descripción	Dado un problema y un dominio, el *script* deberá ser capaz de proporcionar un conjunto de variables *relevantes* de acuerdo con las investigaciones realizadas, con el fin de resolver dicho problema en el dominio en cuestión.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☐ Interno ☑ Daniel Borrajo
Responsable	José González Barroso		

Tabla 106. Requisito de usuario 1 – DI

RU-DI2			
Nombre	Ejecutar iPDB modificado con las *variables relevantes* obtenidas.		
Descripción	Se ejecutará iPDB automáticamente en el dominio y problema en cuestión, pero teniendo en cuenta tanto las variables relevantes como los parámetros introducidos para Ipdb, modificando el improvement en las PDBs con estas variables.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 107. Requisito de usuario 2 – DI

3.3.8.1.2. Casos de uso

Al igual que en los requisitos de usuario a alto nivel, el número casos de uso es mínimo, ya que solo se puede hacer una cosa con el *script*. Para ser exactos, el número de casos de uso es de tan solo 1.

CU-DI1	
Nombre	Resolver problema en cualquier dominio.
Descripción	Dado un problema y un dominio, resolverlo dicho problema.
Actor/es	Cualquier usuario.
Requisito/s relacionados/s	Todos los requisitos de usuario de alto nivel y todos los requisitos funcionales a bajo nivel relacionados con la etiqueta DI (independiente del domino).
Precondiciones	Haber accedido a una consola de comandos y al directorio donde se encuentra el *script AlgGeneral* y el dominio y el problema a resolver.
Postcondiciones	Contemplar las trazas de ejecución del algoritmo sobre ese dominio y problema, el output.sas y el plan (en caso de encontrar solución).

Tabla 108. Caso de uso – DI

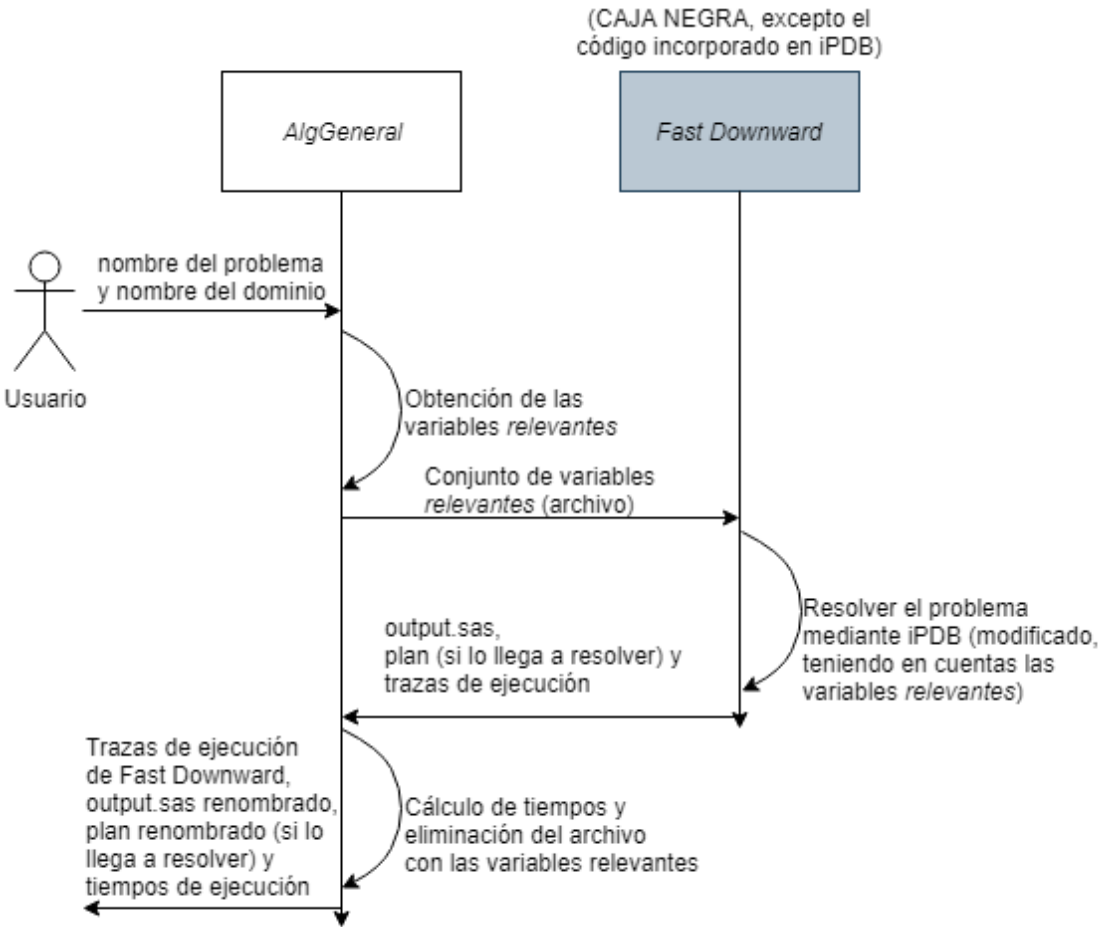

Ilustración 6. Diagrama de casos de uso - DI

3.3.8.1.3. Manual de usuario

Para utilizar el algoritmo tan solo hay que acceder al directorio donde se encuentra, junto al dominio y el problema a resolver y escribir por consola el siguiente comando:

python AlgGeneral.py <dominio> <problema> <parámetros adicionales iPDB>

Donde *<dominio>* es el archivo del dominio a resolver, *<problema>* es el archivo del problema a resolver y *<parámetros adicionales iPDB>* la lista de parámetros que introducirle a *iPDB* (por ejemplo: *collection_max_size*=10000000, *num_samples*=2000). Este último es opcional.

3.3.8.1.4. Requisitos funcionales a bajo nivel

A continuación, se muestran los requisitos funcionales del sistema a bajo nivel:

RF– DI01			
Nombre	Recibir el problema a resolver – Independiente del dominio.		
Descripción	Recibir el nombre del problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 109. Requisito de software 1 – DI

RF– DI02			
Nombre	Recibir el dominio – Independiente del dominio.		
Descripción	Recibir el nombre del dominio relacionado con el problema a resolver.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 110. Requisito de software 2 – DI

RF– DI03			
Nombre	Recibir parámetros de *iPDB* – Independiente del dominio.		
Descripción	Recibir los parámetros de *iPDB* por parámetro al ejecutar *AlgGeneral*. Estos se le introducirán a iPDB al ejecutarlo.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 111. Requisito de software 3 – DI

RF– DI04			
Nombre	Crear *output* - Independiente del dominio.		
Descripción	Crear *output* del problema del dominio en cuestión para su posterior interpretación.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 112. Requisito de software 4 – DI

RF– DI05			
Nombre	Identificar predicados meta, junto a su *signo* - Independiente del dominio.		
Descripción	Abrir el fichero del problema e identificar los predicados meta, junto a su signo *not* si lo tienen.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 113. Requisito de software 5 – DI

RF– DI06			
Nombre	Identificar *predicados relevantes* - Independiente del dominio.		
Descripción	Abrir el fichero del dominio e identificar los predicados meta (junto a su signo *not* si lo tienen) en los efectos de las acciones. En caso de identificar algún predicado meta, colocar todos los predicados de las precondiciones en una lista de predicados *relevantes* en caso de ser la primera acción identificada. Si no se trata de la primera acción que tiene predicados meta en los efectos, recorrer la lista de predicados *relevantes* y eliminar aquellos predicados que no aparecen en las precondiciones de la acción analizada. De esta forma se realiza una intersección entre conjuntos. Introducir directamente como *pedicados relevantes* aquellas variables meta, siempre y cuando no estén ya incluidas en esa lista.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 114. Requisito de software 6 – DI

RF– DI07			
Nombre	Identificar variables *relevantes* - Independiente del dominio.		
Descripción	Abrir el fichero *output.sas*. En primer lugar, crear una lista con tantos ceros como variables haya. Después, seguir leyendo el fichero y, por cada variable que contenga una instancia de un *predicado relevante*, cambiar el cero correspondiente a la variable en cuestión de la lista creada por un uno.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 115. Requisito de software 7 – DI

RF– DI08			
Nombre	Crear fichero con las *variables relevantes* - Independiente del dominio.		
Descripción	Crear fichero auxiliar a partir de la lista creada. Este contiene el número de variables del *output.sas*, seguido de una línea por cada variable, indicando con un 1 si es relevante y con un 0 en caso contrario.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 116. Requisito de software 8 – DI

RF– DI09			
Nombre	Llamar al planificador - Independiente del dominio.		
Descripción	Ejecuta el algoritmo *iPDB* sobre el dominio y el problema en cuestión y con los parámetros introducidos en *adicional*. Su versión modificada se ejecutará en su lugar, ya existe el fichero con las variables relevantes.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 117. Requisito de software 9 – DI

RF– DI10			
Nombre	Leer fichero de *variables relevantes* y guardar su información - Independiente del dominio.		
Descripción	Si el fichero de las variables *relevantes* existe, se abre. Tras esto, se crea una estructura de datos para almacenar el contenido haciendo uso del número de variables. Posteriormente, se introduce el valor binario correspondiente a cada una de las variables. Finalmente, se cierra el fichero.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 118. Requisito de software 10 – DI

RF– DI11			
Nombre	Incrementar *improvement* - Independiente del dominio.		
Descripción	Tras calcular el *improvement*, se comprueba si este supera el valor mínimo, siempre y cuando exista el fichero de variables *relevantes*. En caso de que exista, y supere el valor mínimo se crea una variable llamada *factorMultiplicativo*. Posteriormente, se observan las variables de la PDB en cuestión junto a su valor en la estructura de datos con las *variables relevantes*, sumando dicho valor a *factorMultiplicativo*. Si una variable es *relevante*, incrementará el valor de *factorMultiplicativo* una unidad. Tras esto, se actualiza el improvement multiplicándolo por *factorMultiplicativo*.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 119. Requisito de software 11 – DI

RF– DI12			
Nombre	Guardar *output* - Independiente del dominio.		
Descripción	Renombrar el archivo *output*, con el fin de que no se sobrescriba con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 120. Requisito de software 12 – DI

RF– DI13			
Nombre	Borrar fichero con las *variables relevantes* - Independiente del dominio.		
Descripción	Borrar el fichero con las *variables relevantes* tras haberse ejecutado el planificador.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 121. Requisito de software 13 – DI

RF– DI14			
Nombre	Guardar plan solución - Independiente del dominio.		
Descripción	Renombrar el archivo *sas* que contiene el plan (si se llega a resolver el problema), con el fin de que no se sobrescriban con las siguientes ejecuciones.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 122. Requisito de software 14 – DI

RF-DI15			
Nombre	Obtener tiempos - Independiente del dominio.		
Descripción	Obtener los tiempos de construcción del fichero *output*, de obtención de las *variables relevantes*, de ejecución del planificador y de guardado de archivos.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 123. Requisito de software 15 – DI

3.3.8.1.5. Requisitos no funcionales

A continuación, se muestran los requisitos no funcionales del sistema:

RNF-DI01			
Nombre	Aceptación de *AlgGeneral*.		
Descripción	Aceptar *AlgGeneral* como algoritmo válido siempre y cuando supere o alcance la calidad de *iPDB*.		
Claridad	☐ Alta ☑ Media ☐ Baja	**Verificabilidad**	☐ Alta ☑ Media ☐ Baja
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☑ Alta ☐ Media ☐ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 124. Requisito no funcional 1 – DI

RNF-DI02			
Nombre	Plan de pruebas - *AlgGeneral*.		
Descripción	Elegir 5 problemas, uno por cada dominio, y observar la salida esperada y obtenida por *AlgGeneral*, respecto a las variables *relevantes*. Si es la misma, el algoritmo se verificará, y se validará si cumple los requisitos de usuario. Por otra parte, la validación y verificación de *iPDB* modificado se basará en la observación de las trazas de ejecución.		
Prioridad	☑ Alta ☐ Media ☐ Baja	**Necesidad**	☑ Esencial ☐ Deseable ☐ Opcional
Claridad	☑ Alta ☐ Media ☐ Baja	**Verificabilidad**	☐ Alta ☐ Media ☑ Baja
Estabilidad	☑ Si ☐ No	**Fuente**	☑ Interno ☐ Daniel Borrajo
Responsable	José González Barroso		

Tabla 125. Requisito no funcional 2 – DI

3.3.8.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Dado que el número de casos de uso es solo uno y este se relaciona con todos los requisitos de usuario de alto nivel y funcionales a bajo nivel, la matriz asociada se compondría de tan solo una línea.

3.3.8.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

RU/RS	01	02	03	04	05	06	07	08	09	10	11	12	13	14	15
1	X	X		X	X	X	X	X					X		X
2	X	X	X					X	X	X	X	X	X	X	X

Tabla 126. Matriz de trazabilidad – DI

Dado que toda fila y toda columna tiene una cruz, podría decirse que son correctos.

3.3.8.2. Arquitectura del sistema

El número de clases del sistema es solo de 1, por lo que no tiene sentido construir una arquitectura de un sistema basado en un único fichero y en una sola clase. Este punto no aplica. Observando el diagrama del caso de uso 3.3.8.1.2 Casos de uso y en la descripción de funciones implementadas 3.3.8.3 Funciones implementadas se puede entender el código del *script* implementado.

3.3.8.3. Funciones implementadas

En este punto se describen brevemente las funciones implementadas:

- ***crearOutput(dominio, problema)***: crea el archivo *output.sas* del problema.

- ***obtencionVariablesRelevantes(dominio, problema)***: esta función identifica las variables relevantes y crea un fichero adicional para que, al llamar a *iPDB*, este lo identifique y proceda a incentivar las PDBs con dichas variables en el proceso de creación de patrones. A continuación, se muestra el procedimiento de actuación del algoritmo:

  - ⇨ Abrir el fichero del problema e identificar los predicados meta, junto a su signo *not* si lo tienen.
  - ⇨ Abrir el fichero del dominio e identificar los predicados meta (junto a su signo *not* si lo tienen) en los efectos de las acciones. En caso de identificar algún predicado meta, colocar todos los predicados de las precondiciones en una lista de *predicados relevantes* en caso de ser la primera acción identificada. Si no se trata de la primera acción que tiene predicados meta en los efectos, recorrer la

lista de *predicados relevantes* y eliminar aquellos predicados que no aparecen en las precondiciones de la acción analizada. De esta forma se realiza una intersección entre conjuntos.

- ⇨ Introducir directamente como *pedicados relevantes* aquellas variables meta, siempre y cuando no estén ya incluidas en esa lista.
- ⇨ Abrir el fichero *output.sas*. En primer lugar, crear una lista con tantos ceros como variables haya. Después, seguir leyendo el fichero y, por cada variable que contenga una instancia de un predicado relevante, cambiar el cero correspondiente a la variable en cuestión de la lista creada por un uno.
- ⇨ Crear fichero auxiliar a partir de la lista creada. Este contiene el número de variables del *output.sas*, seguido de una línea por cada variable, indicando con un 1 si es relevante y con un 0 en caso contrario.

- ***ejecucionPlanificador(dominio, problema, adicional)***: ejecuta el algoritmo *iPDB* sobre el dominio y el problema en cuestión y con los parámetros introducidos en *adicional*. Su versión modificada se ejecutará en su lugar, ya existe el fichero con las variables relevantes.

- ***guardarArchivos(dominio, problema)***: renombra el archivo *sas* que contiene el plan y el archivo *output*, con el fin de que no se sobrescriban con las siguientes ejecuciones. Además, se borra el fichero auxiliar que contiene las variables *relevantes*.

- ***main()***: ejecuta las funciones *crearOutput*, *obtencionVariablesRelevantes*, *ejecucionPlanificador* y *guardarArchivos*, en ese orden. Además, muestra los tiempos que el algoritmo ha tardado en cada una de las funciones.

3.3.9 Modificaciones en *iPDB*

A continuación, se comentarán las modificaciones realizadas sobre el algoritmo implementado *iPDB*, de *Fast Downward*.

3.3.9.1. Requisitos y casos de uso

3.3.9.1.1. Requisitos de usuario a alto nivel

Los requisitos de usuario pertenecientes a esta modificación se encuentran incluidos en los requisitos de usuario del algoritmo independiente del dominio 3.3.8.1.1. Requisitos de usuario a alto nivel.

3.3.9.1.2. Casos de uso

El usuario no puede acceder a utilizar este algoritmo modificado si no es mediante *AlgGeneral*. Dicho esto, este punto no aplica.

3.3.9.1.3. Manual de usuario

El usuario no puede acceder a utilizar este algoritmo modificado si no es mediante *AlgGeneral*. Dicho esto, este punto no aplica.

3.3.9.1.4. Requisitos funcionales a bajo nivel

Los requisitos funcionales software pertenecientes a esta modificación se encuentran incluidos en los requisitos funcionales *software* del algoritmo independiente del dominio 3.3.9.1.4 Requisitos funcionales a bajo nivel.

3.3.9.1.5. Requisitos no funcionales

Los requisitos no funcionales de *iPDB* modificado están incluidos en los requisitos no funcionales de *AlgGeneral* 3.3.9.1.5 Requisitos no funcinales.

3.3.9.1.6. Matriz de trazabilidad entre requisitos y casos de uso

Al no existir casos de uso, este punto no aplica.

3.3.9.1.7. Matriz de trazabilidad entre requisitos de usuario y requisitos funcionales de *software*

Dado que no existen requisitos de usuario exclusivos para *iPDB* modificado, este punto no aplica.

3.3.9.2. Arquitectura del sistema

Esta modificación no varía la arquitectura del sistema del algoritmo original, por lo que no aplica.

3.3.9.3. Funciones implementadas

En este punto se describen solamente los cambios realizados en las funciones:

- ***hill_climbing (…)***: los cambios en esta función se detallan a continuación:

  - ⇨ Antes que nada, si el fichero de las variables *relevantes* existe, se abre. Tras esto, se crea una estructura de datos para almacenar el contenido haciendo uso del número de variables. Posteriormente, se introduce el valor binario correspondiente a cada una de las variables. Finalmente, se cierra el fichero.
  - ⇨ Se añade un parámetro más a la llamada de la función *find_best_improving_pdb*.
  - ⇨ Cuando se detiene el proceso de creación de PDBs, se borra la estructura de datos con las *variables relevantes*.

- ***find_best_improving_pdb (…)***: los cambios en esta función se detallan a continuación:

  - ⇨ Se añade un parámetro más a la función.
  - ⇨ Tras calcular el *improvement*, se introduce un condicional para ver si este supera el valor mínimo, siempre y cuando exista el fichero de *variables relevantes*. En caso de que exista y supere el valor mínimo, se crea una variable llamada *factorMultiplicativo*. Posteriormente, se observan las variables de la PDB en cuestión junto a su valor en la estructura de datos con las variables *relevantes*, sumando dicho valor a *factorMultiplicativo*. Si una variable es *relevante*, incrementará el valor de *factorMultiplicativo* una unidad. Tras esto, se actualiza el *improvement* multiplicándolo por *factorMultiplicativo*.

3.3.10 *Scripts* para automatizar las llamadas a algoritmos

Para automatizar las llamadas a los algoritmos implementados *AlgSnake*, *AlgTermes*, *AlgHiking*, *AlgSpider*, *AlgTetris* e *iPDB* modificado, al igual que a los algoritmos que proporciona *Fast Downward*, es necesario implementar un *script*. Este tan solo hace uso de la librería de *Python commands*, introduciéndole por parámetro a la función getoutput el comando que se desea automatizar, por ejemplo, python *AlgSnake.py 01*.

Además, se renombran los ficheros *output.sas*, el plan solución (en caso de que exista) para que no se sobrescriban con la siguiente ejecución. También, como resulta evidente, se recogen las trazas de la ejecución para analizarlas posteriormente.

Estos *scripts*, al ser tan sencillos y tan solo servir de ayuda para agilizar el desarrollo del proyecto, no requiere de una especificación de requisitos.

3.3.11 Otros *scripts* auxiliares para agilizar el trabajo

Al igual que los *scripts* de llamadas a algoritmos, estos tan solo sirven de ayuda para agilizar el desarrollo del proyecto. Por tanto, no requiere de una especificación de requisitos.

Estas ayudan a recopilar datos de las trazas de ejecución, ilustrar visualmente el desarrollo del plan solución en los problemas de algunos dominios, comprender el funcionamiento de ciertos algoritmos de *Fast Downward*, etc. De forma resumida, podría decirse que todos estos programas implementados podrían servir para dos posibles cosas:

- Para recopilar información de las trazas de ejecución.

- Para entender mejor los algoritmos (como un soporte para estudiarlos y comprenderlos).

Además, es necesario recalcar que se han introducido impresiones en los algortimos analizados, como *iPDB*, con el fin de poder ver las PDBs podadas, los conjuntos aditivos creados, etc.

SECCIÓN 4. INVESTIGACIÓN

El proceso de investigación radica en 7 partes principales: un estudio previo para elegir que dominios y algoritmos utilizar, después el estudio en cada uno de los 5 dominios elegidos y un último proceso de investigación para elaborar una heurística independiente del dominio.

La metodología que se va a seguir se encuentra redactada al completo en al apartado 3.1.1 Esqueleto de la investigación.

4.1 Estudio previo

La metodología correspondiente a este punto se encuentra redactado en los apartados 3.1.4 Elección de los dominios y 3.1.5 Elección de los algoritmos de estudio.

4.1.1 Selección de dominios

En primer lugar, se escogerán los dominios que se van a estudiar. Este estudio previo es poco intensivo dado que bastará con elegir 5 dominios que funcionen bien con PDBs. Por esa razón, el análisis de cada dominio es superficial. A continuación, se mostrará un conjunto acotado de dominios, indicando por qué razón se han descartado o elegido. La información acerca de los dominios elegidos podrá observarse detalladamente más adelante.

- Dominios del *track clásico* (determinista) de la **IPC de 2018**:

- ⇨ ***Agrícola***: el tiempo de ejecución mediante *iPDB* es demasiado grande, y al acortarlo mediante *max_time*, son pocos los dominios que resuelve. DESCARTADO.
- ⇨ ***Nurikabe***: no puede ejecutarse mediante *iPDB*, dado que tiene expresiones forall en el dominio. DESCARTADO.
- ⇨ ***Organic-synthesis***: en la mayoría de los problemas y dominios no se llega a generar el output.sas, pues tarda muchísimo tiempo. DESCARTADO.
- ⇨ ***Organic-synthesis-split***: son tantas las variables que se generan que el tiempo de ejecución de *iPDB* es demasiado grande. Además, al añadirle un tiempo límite de generación de PDBs, estas no son demasiado acertadas, pues no le ha dado tiempo para generar buenos patrones. DESCARTADO.
- ⇨ ***Settlers***: no puede ejecutarse mediante *iPDB*, dado que tiene expresiones forall en el dominio. DESCARTADO.

- ⇨ ***Snake***: *iPDB* resuelve 13 problemas de 20 (sin cambiar los parámetros). Los *output.sas* contienen muchas variables, pero no un número excesivo. El número de PDBs generadas tiende a ser grande (sin contar las individuales), y contienen bastantes variables. Estas características se cumplen en casi todos los problemas. Por otra parte, observando las metas de los problemas se puede observar que se suelen componer de bastantes parámetros instanciados. ELEGIDO.
- ⇨ ***Spider***: *iPDB* resuelve 16 problemas de 20 (sin cambiar los parámetros). Los *output.sas* contienen más variables que los del dominio *Snake*. El número de PDBs generadas no es tan grande (sin contar las individuales), pero forma bastantes subconjuntos aditivos. Estas PDBs contienen pocas variables. Esto sucede en gran parte de los problemas. Por otro lado, observando las metas de los problemas se puede observar que se suelen componer de bastantes parámetros instanciados. ELEGIDO.
- ⇨ ***Termes***: *iPDB* resuelve 13 problemas de 20 (sin cambiar los parámetros). Los *output.sas* contienen pocas variables. El número de PDBs generadas es pequeño (sin contar las individuales), pero estas contienen bastantes variables. Estas características se cumplen en casi todos los problemas. Por otra parte, observando las metas de los problemas se puede observar que se suelen componer de bastantes parámetros instanciados. ELEGIDO.

- Dominios del *track clásico* (determinista) de la **IPC de 2018**:

- ⇨ ***Barman***: no resuelve ningún problema. O no termina generando las PDBs o, cuando lo hace, no resuelve el problema de búsqueda con dichas PDBs. DESCARTADO.
- ⇨ ***Hiking***: *iPDB* resuelve 13 problemas de 20 (sin cambiar los parámetros). Los output.sas contienen pocas variables. El número de PDBs generadas es muy pequeño y estas contienen pocas variables. Estas características se cumplen en casi todos los problemas. Por otra parte, observando las metas de los problemas se puede observar que se suelen componer de pocos parámetros instanciados. ELEGIDO.
- ⇨ ***Maintenance***: es un dominio demasiado *trivial*, con muy pocos problemas. DESCARTADO.
- ⇨ ***Parking***: las PDBs que genera solo contienen una única variable, por lo que no resulta interesante. DESCARTADO.
- ⇨ ***Tetris***: *iPDB* resuelve 9 problemas de 17 (mediante un tiempo límite establecido en 1 minuto). Los *output.sas* contienen más variables que los del dominio Snake. El número de PDBs generadas tiende a ser muy pequeño (sin contar las individuales), pero contienen muchas variables. Estas características se cumplen en gran parte de los problemas. Por otra parte, observando las metas de los problemas se puede observar que se suelen componer de bastantes parámetros instanciados. ELEGIDO.
- ⇨ ***Visitall***: el dominio solo tiene una única acción. DESCARTADO.

Estos dominios no han sido los únicos que se han analizado brevemente. Sin embargo, no es necesario hacer un análisis exhaustivo, pues solo basta con elegir 5 dominios interesantes. Es posible que se haya descartado algún dominio prometedor, pero no importa. Además, también se quería hacer mostrar que no todos los dominios funcionan bien usando algoritmos que utilicen PDBs.

Los dominios que se han seleccionado son interesantes porque, además de que funcionan bien con PDBs, cada uno de ellos tiene peculiaridades distintas al resto. De esta forma, la heurística independiente creada a partir del estudio de los resultados de estos dominios tenderá a ser más precisa, puesto que se intentará generalizar mediante estos dominios tan dispares. Observando las *tablas 127 y 128*, podrá comprobarse que este estudio inicial ha finalizado exitosamente.

Dominios vs característica	Resuelve > 50% de los problemas	Pocas variables en *output.sas*	Pocas PDBs formadas	Pocas Variables en PDB
Snake	X			
Termes	X	X	X	
Hiking	X	X	X	X
Spider	X		X	X
Tetris	X		X	

Tabla 127. Dominios vs características 1

Dominios vs característica	pocas metas en los problemas	Requiere *max_time*	Suele formar conjuntos aditivos	Las PDBs suelen seguir un patrón *claro*
Snake				X
Termes				X
Hiking	X			X
Spider			X	
Tetris		X	X	

Tabla 128. Dominios vs características 2

4.1.2 Selección de algoritmos

En segundo lugar, se escogerán los algoritmos que se van a utilizar en el estudio. La información aquí expuesta acerca de los algoritmos se puede extraer del estado del arte Sección 2 Estado del arte. Por esa razón, no es necesario realizar un análisis meramente empírico de cada uno de los algoritmos a elegir. En ciertos algoritmos, un breve estudio analítico será suficiente para elegirlos o descartarlos.

En *Fast Downward* se puede observar que los algoritmos que hacen uso de PDBs son: *Canonical PDB*, *Zero-One PDB*, *iPDB* y *Pattern database heuristic*. A continuación, se analiza uno por uno:

- ***Pattern database heuristic***: no es una buena alternativa, pues no permite operar con más de una PDB. Por esa razón queda completamente descartado, pues no es interesante.

- ***iPDB***: este algoritmo es equivalente a usar *Hill climbing* en la obtención de patrones sobre *Canonical PDB*. Puede ser algoritmo interesante para observar los patrones resultantes y para comparar resultados con los otros algoritmos. Por otra parte, los parámetros no parecen ser críticos para experimentar exhaustivamente con estos, sino que sería adecuado hacer uso de ellos en caso de requerirlo. Se analizará a continuación cada uno de ellos:

  ⇨ ***pdb_max_size***: podría resultar interesante incrementar su valor en caso de que alguna PDB no haya aumentado su número de variables (abortando dicha PDB

por alcanzar el límite de número de estados), hasta el punto de poder realizar un buen análisis.

- ⇨ ***collection_max_size***: en caso de que algún dominio tienda a construir muchas PDBs y muy grandes, aumentar este valor podría resultar interesante.
- ⇨ ***num_samples***: si se diese el caso de que en un dominio y en sus problemas se construyesen PDBs muy pequeñas y poco interesantes para generalizar, aumentar mucho este valor ayudaría a incrementar el tamaño de las PDBs. Disminuir este valor no resulta del todo interesante. Quizás valdría la pena en caso de obtener PDBs muy grandes con variables no tan necesarias, pero estas se descartarían tras realizar las observaciones.
- ⇨ ***min_improvement***: al igual que *num_samples*, si se diese el caso de que en un dominio y en sus problemas se construyesen PDBs muy pequeñas y poco interesantes para generalizar, decrementar mucho este valor ayudaría a incrementar el tamaño de las PDBs. En caso de querer tomar esta decisión, se escogerá modificar *min_improvement* en lugar de *num_samples*.
- ⇨ ***max_time***: en caso de que, en un dominio, la mayor parte de sus problemas no se puedan resolver debido al tiempo de generación de PDBs, acotar este valor podría resultar interesante.
- ⇨ ***random_seed***: variar este parámetro no resulta determinante para el estudio.
- ⇨ ***max_time_dominance_pruning***: en caso de que el tiempo dedicado a la poda y a establecer los conjuntos de PDBs aditivas sea excesivamente grande, acotar este valor podría resultar interesante.
- ⇨ ***transform***: variar este parámetro no resulta determinante para el estudio.
- ⇨ ***cache_estimates***: variar este parámetro no resulta determinante para el estudio.

- ***Canonical PDB***: este algoritmo puede ser interesante para observar los patrones resultantes y para comparar resultados con los otros algoritmos. Por otra parte, los parámetros no parecen ser críticos para experimentar exhaustivamente con todos estos, sino que sería adecuado hacer uso de ellos en caso de requerirlo. Sin embargo, se analizará a continuación cada uno de ellos:

- ⇨ ***patterns***: a continuación, se expone cada uno de los posibles valores de estos:
  - o Combo: este método de generación de patrones podría resultar interesante de analizar. Además, se puede introducir un parámetro:
    - Max_states: como lo que interesa de este algoritmo es observar si la inclusión de las metas en una PDB es importante, experimentar con este parámetro no proporcionaría ningún tipo de información extra.
  - o Patrones generados por algoritmos genéticos: debido a que el código fuente no funciona debidamente en ciertos dominios, no se abordará en el proyecto. Suele mandar un mensaje de error cuando los patrones obtenidos contienen muchas variables, y no llega a mostrar ninguna PDB. Se suele quedar estancado sin dar ninguna solución. También, en

ocasiones, muestra un mensaje de error a la hora de podar. Sería muy poco sistemático ejecutarlo solo en ciertos dominios. En trabajos futuros podría considerarse, siempre y cuando se asegure su funcionamiento en todos los dominios con los que se va a trabajar.

- *Hill Climbing*: este algoritmo es el mismo que el de generación de patrones de iPDB.
- Patrones manuales: esta alternativa se utilizará simplemente en el algoritmo a implementar.
- Generación de patrones sistemática: este algoritmo forma toda la combinatoria posible de variables, incluyendo tantas en una PDB como valor indique el parámetro *pattern_max_size*. Por otra parte, el parámetro *only_interesting_patterns* elimina todas las PDBs que tengan variables que no aporten nada nuevo al combinarlas. Dado que el estudio dependiente del dominio se basa en observar cuáles son las variables relevantes, realizar una combinatoria de todas ellas y utilizarlas a la vez no es del todo acertado. De todos modos, hay varias objeciones acerca de este algoritmo de creación de patrones:
  - En el caso de eliminar los patrones innecesarios, todos los restantes son exactamente los patrones que iPDB analiza en su algoritmo, por lo que añadirlos todos sin generalizar sería una mala alternativa.
  - Son tantos los patrones que se generarían que difícilmente cabrían en memoria.
  - Poner un valor demasiado bajo en *pattern_max_size* formaría patrones poco útiles.
  - Poner un valor demasiado alto en *pattern_max_size* formaría tantos patrones que el tiempo de generación de estos tardaría muchísimo tiempo. Sin ir más lejos, de forma experimental se probó ejecutar este algoritmo con un *pattern_max_size* de tamaño 3 en el primer problema del dominio Snake, y tras más de dos horas de ejecución aún no había generado todas las PDBs resultados de la combinatoria.

⇨ ***max_time_dominance_pruning***: en caso de que el tiempo dedicado a la poda y a establecer los conjuntos de PDBs aditivas sea excesivamente grande, acotar este valor podría resultar interesante.
⇨ ***transform***: variar este parámetro no resulta determinante para el estudio.
⇨ ***cache_estimates***: variar este parámetro no resulta determinante para el estudio.

- ***Zero-One PDB***: personalmente, considero que la forma de interpretar las PDBs en este algoritmo es demasiado primitiva y básica. Sin embargo, sería inapropiado descartar el uso de este algoritmo por una cuestión meramente subjetiva. Por otra parte, los parámetros que tiene no parecen ser críticos para experimentar exhaustivamente con

todos estos, sino que sería adecuado hacer uso de ellos en caso de requerirlo. Sin embargo, se analizará a continuación cada uno de ellos:

- ⇨ ***patterns***: a continuación, se expone cada uno de los posibles valores de estos:
  - *Combo*: idem *Canonical PDB*.
  - Patrones generados por algoritmos genéticos: idem *Canonical PDB*.
  - *Hill Climbing*: esta opción, al contrario que en *Canonical PDB*, no se corresponde al algoritmo *iPDB*, debido a la interpretación que se hace de los patrones. Sin embargo, tampoco se ha decidido incorporarlo en la lista de algoritmos para ejecutar. Las razones son las siguientes:
    - Si existe un algoritmo particular que emplea *Hill climbing* y realiza la misma interpretación de las heurísticas que *Canonical PDB*, incita a pensar que sería una mejor alternativa usar esa interpretación de las heurísticas.
    - No realiza ni poda ni conjuntos aditivos, con lo que muchas PDB que se observan son innecesarias.
    - Las PDBs obtenidas antes de las podas son las mismas que en iPDB.
    - El orden de las PDBs es importante, por lo que sería necesario invertir mucho tiempo en probar distintas ordenaciones para observar si es una buena opción.
    - La potencia de la heurística jamás llegará a superar al del algoritmo *iPDB*.
    - La información que proporcione este algoritmo ya la proporciona *iPDB*, y en mayor medida.
  - Patrones manuales: idem *Canonical PDB*.
  - Generación de patrones sistemática: idem *Canonical PDB*.
- ⇨ ***transform***: idem *Canonical PD*B.
- ⇨ ***cache_estimates***: idem *Canonical PDB*.

Dicho esto, y como resumen, los algoritmos elegidos son los siguientes:

- Para la fase de observación:

  - ⇨ *iPDB*, variando *pdb_max_size*, *collection_max_size*, *min_improvement*, *max_time* y/o *max_time_dominance_pruning* en caso necesario.
  - ⇨ *Canonical PDB* usando el algoritmo *combo*, variando *max_time_dominance_pruning* en caso necesario.
  - ⇨ *Zero-One PDB* usando el algoritmo *Combo*.

- Los algoritmos que se implementarán:

  - ⇨ Mediante *Canonical PDB*, el algoritmo para introducir patrones de forma manual, variando *max_time_dominance_pruning* en caso necesario.

⇨ Mediante *Zero-One PDB*, el algoritmo para introducir patrones de forma manual.

- Para la fase de evalución:

⇨ Los mismos algoritmos que en la fase de observación.
⇨ Los mismos algoritmos que en la fase de implementación.

4.2 Dominio *Snake*

4.2.1 Ejecución de los algoritmos

El primer paso es ejecutar los algoritmos seleccionados en el punto 4.1.2 Selección de algoritmos. Los resultados son los siguientes (*tablas 129 – 131 y 193 – 196*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	897.092	1.02942	0.797428
P02	279.163	11.5411	10.3191
P03	361.447	50.0486	49.5196
P04	84.5717	0.363598	0.341426
P05	-	0.147108	0.140635
P06	446.45	14.7138	14.7781
P07	513.355	-	-
P08	-	-	-
P09	912.248	0.555213	0.384601
P10	430.939	0.772374	0.697996
P11	641.723	32.6433	32.759

P12	666.832	-	-
P13	-	-	-
P14	-	-	-
P15	954.978	0.436471	0.513233
P16	709.584	464.115	464.364
P17	-	-	-
P18	-	-	-
P19	-	-	-
P20	1173.33	1.56562	1.54009

Tabla 129. Tiempo total – Snake – Observación

Número nodos expandidos	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	44	7365	7365
P02	5188	736860	736860
P03	1264851	4253977	4254346
P04	13	13	13
P05	-	237	237
P06	333	1019228	1019228
P07	15710618	-	-
P08	-	-	-

P09	21	263	263
P10	43	14043	14043
P11	371	2516478	2516478
P12	9095686	-	-
P13	-	-	-
P14	-	-	-
P15	44	7005	7005
P16	248924	29343913	29343946
P17	-	-	-
P18	-	-	-
P19	-	-	-
P20	31	34880	34880

Tabla 130. Número nodos expandidos – Snake – Observación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	24	24	24
P02	32	32	32
P03	43	43	43
P04	12	12	12
P05	-	17	17

P06	31	31	31
P07	48	-	-
P08	-	-	-
P09	20	20	20
P10	27	27	27
P11	36	36	36
P12	47	-	-
P13	-	-	-
P14	-	-	-
P15	25	25	25
P16	42	42	42
P17	-	-	-
P18	-	-	-
P19	-	-	-
P20	30	30	30

Tabla 131. Coste de la solución – Snake – Observación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.2.2 Observación de resultados

En primer lugar, se justifican los motivos por los cuales no se han introducido ciertos parámetros:

- ***pdb_max_size***: las PDBs contienen muchas variables (en torno a 14 para ser exactos), y se puede hacer un análisis adecuado de estas.

- ***collection_max_size***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente. De todos modos, el algoritmo de creación de PDBs interrumpía su proceso debido a que ninguna nueva PDB superaba el min_improvement.

- ***min_improvement***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente para hacer un buen análisis.

- ***max_time***: en tan solo un problema sucedió de que el tiempo de generación de las PDBs superó las expectativas y no pudo dar comienzo la búsqueda del plan solución.

- ***max_time_dominance_pruning*** **(en *iPDB*)**: el tiempo de poda es muy pequeño.

- ***max_time_dominance_pruning*** **(en *Canonical PDB*)**: el tiempo de poda es muy pequeño.

En *iPDB*, el tiempo de generación de las PDBs es muy grande respecto a los otros dos algoritmos. Pero el tiempo de búsqueda es mucho menor, al igual que el número de nodos expandidos (cuando el problema se resuelve). Por otra parte, el tiempo de poda es despreciable en *iPDB* y en *Canonical PDB*. En resumidas cuentas, el tiempo que iPDB ahorra en el proceso de búsqueda no llega a compensar el tiempo empleado en generar las PDBs. Respecto a la memoria, las PDBs que este primer algoritmo construye utilizan mucha memoria, al contrario que los otros dos algoritmos.

Además, entre *Canonical PDB* y *Zero-One PDB* las diferencias no son muy notables. Se observa cómo, en ocasiones, el tiempo de *Canonical PDB* puede llegar a ser ligeramente superior a *Zero-One PDB*, pero el número de nodos expandido es siempre mayor o igual. Se puede llegar a la conclusión de que, mediante *Canonical PDB* la heurística es más informada que en *Zero-One*, a pesar de que los patrones sean los mismos. Pero el tiempo puede llegar a ser mayor (aunque no tanto) debido a que en este primero son necesarias más consultas a tablas, más tiempo de poda y de generación de conjuntos aditivos, etc. La cantidad de memoria utilizada es similar, y el número de problemas resueltos, el mismo.

También se puede ver cómo *iPDB* resuelve dos problemas que los otros dos algoritmos no son capaces de resolver. A su vez, estos dos algoritmos resuelven el problema en el que *iPDB* no había iniciado aún el proceso de búsqueda del plan. Y, como resulta evidente, los costes de los planes generados son exactamente los mismos. Esto no implica que los planes sean iguales, pues puede darse el caso de que exista más de un plan meta con el mismo coste.

Tras este análisis, se puede observar cómo *iPDB*, a pesar del gran consumo de tiempo y la memoria, es la mejor alternativa hasta ahora, ya que resuelve más problemas. Implementar un algoritmo basándose en el estudio de *iPDB* sobre este dominio es más interesante, pues ofrece un conjunto de PDBs productos de un proceso de búsqueda. Podría encontrarse patrones de comportamiento interesantes si se analizasen las PDBs en cuestión.

Las características observadas de cada problema donde el algoritmo *iPDB* ha actuado, realizando previamente la poda y la creación de conjuntos aditivos, son las siguientes:

- Problema 1: las PDBs creadas ocupan mucha memoria. Aparecen bastantes patrones y con muchas variables. En todas las PDBs aparecen la cabeza de la serpiente, el *spawn* y el primer punto de comida. Las variables *nextsnake* y *tailsnake* no aparecen. Aparecen como PDBs de una sola variable los 5 puntos de comida iniciales. Se repiten siempre las variables meta centrales - finales. Algunos *blocked* que salen coinciden con metas, otros no, salen *blocked* que no coinciden con nada. Creo que no se puede generalizar con *blocked*, no cumplen un criterio general. RESUELTO.

- Problema 2: lo mismo en el problema 1 pero el primer punto de comida no siempre aparece en las PDBs y no se repiten siempre los puntos de comida meta centrales – finales, ya que dichos puntos de comida parecen ser seleccionados de forma aleatoria. Los *blocked* aparecen en menos ocasiones. Aparecen algunos puntos de comida iniciales en los patrones *largos* y repetidos, y los patrones individuales a veces tienen puntos de comida (aleatorios). RESUELTO.

- Problema 3: lo mismo que problema 1, pero solo aparece un *blocked*. Además, se repiten variables de comida centrales. RESUELTO.

- Problema 4: cuando hay pocos puntos de comida podría meter hasta la cola de la serpiente, no hay patrones individuales, hay pocos patrones y las PDBs no ocupan tanto como las anteriores. Creo que pasan cosas poco usuales porque es un problema aparentemente sencillo. RESUELTO.

- Problema 5: no han acabado de crearse las PDBs.

- Problema 6: nada común que no se haya visto anteriormente. RESUELTO.

- Problema 7: nada común que no se haya visto anteriormente. RESUELTO.

- Problema 8: nada común que no se haya visto anteriormente.

- Problema 9: problema muy sencillo, mete algunas variables de *isblocked*, no aparecen PDBs individuales… Es decir, aparecen cosas poco usuales, como en el problema 4. RESUELTO.

- Problema 10: problema medio sencillo, mete algunos *blocked*, no tiene PDBs *individuales*, los puntos de comida iniciales no aparecen todos repetidos, pero el último patrón los tiene todos. RESUELTO.

- Problema 11: nada común que no se haya visto anteriormente. RESUELTO.

- Problema 12: nada común que no se haya visto anteriormente. RESUELTO.

- Problema 13: nada común que no se haya visto anteriormente.

- Problema 14: nada común que no se haya visto anteriormente

- Problema 15: Problema sencillo con *ispawn* en la variable número 0 (*ispawn*). Problema aparentemente medio sencillo, mete algunos *blocked*, no tiene individuales, los puntos de comida iniciales no aparecen todos repetidos, pero el último patrón los tiene todos. RESUELTO.

- Problema 16: nada común que no se haya visto anteriormente. RESUELTO.

- Problema 17: nada común que no se haya visto anteriormente. Problemas complicados: muchos patrones independientes.

- Problema 18: nada común que no se haya visto anteriormente. Problemas complicados: muchos patrones independientes.

- Problema 19: nada común que no se haya visto anteriormente. Problemas complicados: muchos patrones independientes.

- Problema 20: nada común que no se haya visto anteriormente. RESUELTO.

Llegados hasta aquí, y tras contemplar las PDBs de cada problema, se podría generalizar. Para ello, se tenderá a observar con más detalle aquellos problemas resueltos. Las variables *blocked* aparecen cuando los problemas son más sencillos, pero los que siempre o casi siempre suelen salir son variables de metas, la cabeza y el *ispawn*.

Se ha pensado, por tanto, en implementar un algoritmo que cree PDBs con las siguientes características:

En primer lugar, construir en torno a 9 PDBs *largas*, aproximadamente el número de PDBs *largas* que el algoritmo ofrece finalmente. Todas tienen la cabeza de la serpiente, el *ispawn* y después puntos meta aleatorios. El número de variables por PDB *larga* es de 14 aproximadamente, también de acuerdo con las PDBs que *iPDB* muestra en su algoritmo. Además, entre todos los patrones deberían estar todas las variables meta.

Además, poner todos los puntos de comida iniciales como PDBs de una sola variable. Primero se colocan los patrones más grandes, y luego los patrones individuales, con el fin de favorecer a *Zero-One PDB*, aunque para *Canonical PDB* no sea relevante la ordenación.

4.2.3 Implementación y ejecución del algoritmo

En este punto se implementará el algoritmo de acorde a la generalización realizada en el anterior punto. El *script* implementado se comenta en el punto 3.3.3 *Script* de algoritmo para Snake.

Los resultados, dado que se van a comparar posteriormente con el resto de los algoritmos, se muestran en conjunto. Es importante enfatizar que, aunque se muestren dos algoritmos implementados (uno con *Canonical PDB* y otro con *Zero-One PDB*), la realidad es que se tratan del mismo, pero introduciendo su resultado en el modo de introducción de patrones manual de *Canonical* Y *Zero-One* respectivamente (*tablas 132 – 134 y 193 – 196*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSnake (Canonical PDB)*	*AlgSnake (Zero-One PDB)*
P01	897.092	1.02942	0.797428	5.80153393	6.54691393
P02	279.163	11.5411	10.3191	9.94346408	9.61713408
P03	361.447	50.0486	49.5196	15.90446882	35.83066882
P04	84.5717	0.363598	0.341426	0.23493203	0.18065803
P05	-	0.147108	0.140635	0.66840707	0.64387107
P06	446.45	14.7138	14.7781	13.21105392	14.03365392
P07	513.355	-	-	54.68040521	215.88250521
P08	-	-	-	659.32719686	-
P09	912.248	0.555213	0.384601	0.48888902	0.59463902

P10	430.939	0.772374	0.697996	7.01122390	5.52365390
P11	641.723	32.6433	32.759	15.66071493	16.62101493
P12	666.832	-	-	51.33190990	490.09390990
P13	-	-	-	-	-
P14	-	-	-	-	-
P15	954.978	0.436471	0.513233	2.79343696	2.37198696
P16	709.584	464.115	464.364	29.22603607	63.00313607
P17	-	-	-	634.81734115	-
P18	-	-	-	-	-
P19	-	-	-	-	-
P20	1173.33	1.56562	1.54009	10.23564088	12.28974088

Tabla 132. Tiempo total – Snake – Evaluación

Número nodos expandidos	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgSnake (Canonical PDB)***	***AlgSnake (Zero-One PDB)***
P01	44	7365	7365	50	327
P02	5188	736860	736860	6562	42291
P03	1264851	4253977	4254346	400056	2276013
P04	13	13	13	13	40
P05	-	237	237	18	136
P06	333	1019228	1019228	1123	61090

P07	15710618	-	-	2757587	14755563
P08	-	-	-	44679351	-
P09	21	263	263	21	53
P10	43	14043	14043	62	64
P11	371	2516478	2516478	395	14122
P12	9095686	-	-	1876776	30298089
P13	-	-	-	-	-
P14	-	-	-	-	-
P15	44	7005	7005	46	647
P16	248924	29343913	29343946	52478	2201015
P17	-	-	-	33907904	-
P18	-	-	-	-	-
P19	-	-	-	-	-
P20	31	34880	34880	31	4199

Tabla 133. Número nodos expandidos – Snake – Evaluación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSnake (Canonical PDB)*	*AlgSnake (Zero-One PDB)*
P01	24	24	24	24	24
P02	32	32	32	32	32
P03	43	43	43	43	43

P04	12	12	12	12	12
P05	-	17	17	17	17
P06	31	31	31	31	31
P07	48	-	-	48	48
P08	-	-	-	58	-
P09	20	20	20	20	20
P10	27	27	27	27	27
P11	36	36	36	36	36
P12	47	-	-	47	47
P13	-	-	-	-	-
P14	-	-	-	-	-
P15	25	25	25	25	25
P16	42	42	42	42	42
P17	-	-	-	62	-
P18	-	-	-	-	-
P19	-	-	-	-	-
P20	30	30	30	30	30

Tabla 134. Coste de la solución – Snake – Evaluación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.2.4 Pruebas del algoritmo

Estas pruebas consisten en seleccionar 3 problemas. Es favorable que ofrezcan resultados diferentes entre sí, para probar mejor el algoritmo. Se compara la salida esperada con la obtenida, y si coinciden, se podría verificar su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validaría.

Se van a observar los problemas 1, 10 y 20. Un problema fácil, otro intermedio y otro complicado.

- El algoritmo implementado sobre el problema 1 muestra las siguientes PDBs:

[88, 107, 108, 110, 112, 113, 114, 115, 117, 118, 119, 120, 121, 122]

[88, 107, 108, 112, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122]

[88, 107, 108, 110, 111, 112, 113, 114, 115, 116, 118, 119, 121, 122]

[88, 107, 109, 110, 112, 114, 115, 116, 117, 118, 119, 120, 121, 122]

[88, 107, 108, 109, 113, 114, 115, 116, 117, 118, 119, 120, 121, 122]

[88, 107, 108, 109, 112, 113, 115, 116, 117, 118, 119, 120, 121, 122]

[88, 107, 108, 109, 111, 112, 113, 114, 115, 116, 119, 120, 121, 122]

[88, 107, 109, 110, 111, 112, 113, 115, 116, 117, 119, 120, 121, 122]

[108]

[109]

[110]

[111]

[112]

88: *ispawn* (siguiente punto de comida que aparecerá).

107: cabeza de la serpiente.

108 – 112: puntos de comida iniciales.

113 – 122: el resto de los puntos de comida.

- El resto de las pruebas se encuentran en la Sección 9 Anexo.

Tal y como se puede observar, el número de PDBs *largas* es menor igual que 9, pues las repetidas se eliminan. El número de variables es, aproximadamente 14. Todas estas PDBs tienen la cabeza de la serpiente, el *ispawn* y después puntos meta aleatorios. Los puntos de comida iniciales aparecen también como PDBs independientes. Entre todos

los patrones están todos los puntos de comida. Y, por último, el orden de las PDBs está de mayor a menor.

Dicho esto, el algoritmo funciona correctamente. Queda, por tanto, verificado y validado, pues lo importante del algoritmo, dado que es sencillo, son las PDBs que construye. Dadas las características del *Script*, un análisis más exhaustivo relacionado con una metodología más dura no tendría sentido.

4.2.5 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con el resto de los algoritmos.

En primer lugar, respecto al tiempo de generación de las PDBs, el algoritmo implementado no requiere de mucho tiempo. En el peor de los casos no llega a los 28 segundos, mucho menos que en *iPDB* pero más que en los algoritmos *combo*. Tanto en *Zero-One* como en *Canonical* los resultados son parecidos. Esto no se debe al algoritmo de generación de PDBs en sí, sino a la construcción de los grafos y tablas necesarias relacionadas con esos patrones. Por otra parte, los tiempos de poda y obtención de los conjuntos aditivos es despreciable.

Respecto al tiempo de búsqueda, el algoritmo implementado mediante *Canonical PDB* es el que ofrece mejores resultados, después de *iPDB*. Sin embargo, si se utiliza *Zero-One*, *iPDB* lo supera. Dicho esto, el algoritmo que menos tiempo requiere es el recién implementado, dándole esas PDBs mediante introducción manual al algoritmo *Canonical PDB*. A pesar de ello, cuando los problemas son demasiado sencillos, el tiempo que *AlgSnake* gasta en construir las PDBs no llega a equilibrarse con el tiempo ahorrado en la ejecución del planificador, más aún cuando se compara con los algoritmos *combo*.

Respecto a la memoria utilizada, este nuevo algoritmo no supera a *iPDB*. Por otra parte, *AlgSnake* en Canonical emplea poca más memoria que en *Zero-One* cuando se trata de problemas sencillos, pero las tornas se invierten cuando el problema gana dificultad, pues requiere de más tiempo de ejecución. Además, en los problemas extremadamente sencillos, los algoritmos *combo* son los que menos memoria requieren.

Respecto al número de nodos expandidos, dato que refleja la calidad de la heurística, *iPDB* le gana terreno a *AlgSnake* con *Canonical* en los problemas que no requerían de un excesivo tiempo de ejecución. Sin embargo, en problemas más complicados, la heurística implementada es más potente.

Por último, el coste de la solución de todos los algoritmos es el mismo, como cabría esperar. Además, el *AlgSnake* en *Canonical* es el que más problemas resuelve, seguido

de *AlgSnake* en *Zero-One*, *iPDB*, y, por último, los algoritmos *combo*. *AlgSnake* en *Zero-One* solo resuelve un problema más que *iPDB*, y es este en el que *iPDB* se quedaba estancado creando los patrones. Esto no significa que este primero sea más potente que *iPDB*, que no lo es. La verdad es que, para resolver más problemas, el salto de calidad que un algoritmo debe hacer es muy grande, y aun siendo *iPDB* mejor que *AlgSnake* en *Zero-One*, no llega a tener ese gran salto de calidad que el algoritmo *AlgSnake* en *Canonical* sí que tiene.

4.2.6 Extracción de conclusiones

Tras observar la anterior evalución se pueden extraer varias conclusiones respecto a este dominio. En este apartado tan solo se realizarán aquellas más generales.

Cuando los problemas son extremadamente sencillos, bastará con utilizar alguno de los algoritmos combo, sobre todo el de *Zero-One*, para ahorrarnos el proceso de generación de una heurística ligeramente más informada. Sin embargo, cuando la complejidad empieza a ascender más, utilizar *AlgSnake* con *Canonical* es lo mejor. *iPDB* construye mejores heurísticas en problemas sencillos o de dificultad intermedia, pero *AlgSnake* en *Canonical*, al generalizar, es más potente en problemas complejos. De todos modos, esta pequeña diferencia que *iPDB* gana en estos problemas, la pierde al utilizar un tiempo excesivo para la generación del conjunto de patrones.

En resumidas cuentas, el mejor algoritmo es el que se ha implementado, es decir, *AlgSnake* mediante *Canonical*. Resuelve más problemas, y en aquellos sencillos, tampoco es tanta la diferencia temporal respecto a los otros algoritmos.

4.3 Dominio *Termes*

4.3.1 Ejecución de los algoritmos

El primer paso es ejecutar los algoritmos seleccionados en el punto 4.1.2 Selección de algoritmos. Los resultados son los siguientes (*tablas 135 – 137 y 197 – 200*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	3.60109	0.698697	0.671691

P02	3.58253	3.78121	3.82217
P03	4.85755	17.0263	16.9636
P04	7.34706	34.8623	34.379
P05	271.911	-	-
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	5.7526	11.0482	10.788
P12	3.60016	2.2896	2.27298
P13	58.1779	78.4104	77.4558
P14	11.7071	236.067	229.375
P15	-	-	-
P16	-	-	-
P17	81.8209	188.883	145.432
P18	15.6232	46.6441	32.7036
P19	27.0189	91.109	74.7636
P20	126.88	328.406	243.166

Tabla 135. Tiempo total – Termes – Observación

Número nodos expandidos	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	4776	47277	47286
P02	17874	995688	995848
P03	145662	4980609	4989089
P04	748085	10104292	10105099
P05	74493127	-	-
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	219376	3105725	3105766
P12	52374	533081	533241
P13	14958320	23667345	23672375
P14	1953682	69375311	69387188
P15	-	-	-
P16	-	-	-
P17	22443585	45661696	45661896
P18	3403414	9807649	9807890

P19	6027955	22591891	22591922
P20	34655820	74591971	74592951

Tabla 136. Número nodos expandidos – Termes – Observación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	36	36	36
P02	54	54	54
P03	68	68	68
P04	80	80	80
P05	132	-	-
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	66	66	66
P12	46	46	46
P13	92	92	92
P14	104	104	104
P15	-	-	-

P16	-	-	-
P17	116	116	116
P18	76	76	76
P19	94	94	94
P20	106	106	106

Tabla 137. Coste de la solución – Termes – Observación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.3.2 Observación de resultados

En primer lugar, se justifican los motivos por los cuales no se han introducido ciertos parámetros:

- ***pdb_max_size***: las PDBs contienen muchas variables (aproximadamente unas 7, de entre las 13 que suelen aparecer en los archivos *output*), y se puede hacer un análisis adecuado de estas.

- ***collection_max_size***: no es necesario incrementar el tamaño de la colección, pues el número de variables totales no es muy elevado. De todos modos, el algoritmo de creación de PDBs interrumpía su proceso debido a que ninguna nueva PDB superaba el min_improvement.

- ***min_improvement***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente para hacer un buen análisis, debido a la poca cantidad de variables que hay. Igualmente, el algoritmo de generación de PDBs interrumpía su proceso puesto que no se obtenía una nueva PDB con un improvement positivo.

- ***max_time***: el tiempo de generación de las PDBs es, de por sí, muy pequeño.

- ***max_time_dominance_pruning*** **(en *iPDB*)**: el tiempo de poda es muy pequeño.

- ***max_time_dominance_pruning*** **(en *Canonical PDB*)**: el tiempo de poda es muy pequeño.

En *iPDB*, el tiempo de generación de las PDBs es mayor que los otros dos algoritmos, aunque no supera los 5 segundos. Pero el tiempo de búsqueda es muchísimo menor, al igual que el número de nodos expandidos y la cantidad de memoria utilizada. Por otra parte, el tiempo de poda es despreciable en *iPDB* y en *Canonical PDB*. En resumidas cuentas, el tiempo que *iPDB* ahorra en el proceso de búsqueda se compensa con el tiempo empleado en generar las PDBs.

Además, entre *Canonical PD*B y *Zero-One PDB* las diferencias no son muy notables. Se observa cómo el tiempo de *Canonical PDB* puede llegar a ser ligeramente superior a *Zero-One PDB*, pero el número de nodos expandido es siempre mayor o igual. Se puede llegar a la conclusión de que, mediante *Canonical PDB* la heurística es más informada que en *Zero-One*, a pesar de que los patrones sean los mismos. Pero el tiempo puede llegar a ser mayor debido a que en este primero es necesario más consultas a tablas, tiempo de poda y de generación de conjuntos aditivos, etc.; pero no con una gran diferencia. La cantidad de memoria utilizada es similar, y el número de problemas resueltos, el mismo.

También se puede ver cómo *iPDB* resuelve un problema más que los otros dos algoritmos. Y, como resulta evidente, los costes de los planes generados son exactamente los mismos. Esto no implica que los planes sean iguales, pues puede darse el caso de que exista más de un plan meta con el mismo coste.

Tras este análisis, se puede observar cómo *iPDB*, de nuevo, es la mejor alternativa hasta ahora, ya que resuelve más problemas, y con menos tiempo y menos memoria que los otros dos algoritmos. Implementar un algoritmo que supere esta vez a *iPDB* es muy complicado. Implementar un algoritmo basándose en el estudio de *iPDB* sobre este dominio es más interesante, pues ofrece un conjunto de PDBs productos de un proceso de búsqueda mediante *Hill climbing*. Podrían encontrarse patrones de comportamiento interesantes si se analizasen las PDBs en cuestión.

Las características observadas de cada problema donde el algoritmo *iPDB* ha actuado, realizando previamente la poda y la creación de conjuntos aditivos, son las siguientes:

- Problema 1: Los patrones *grandes* (solo 2 y compuestos aproximadamente por 10 variables) tienen la posición del robot, aparecen casi todas las variables excepto la 5 (una posición marginada) y la 10 (otra posición marginada, aunque el robot construye y pasa por ahí). Las PDBs individuales tienen todas excepto la variable 12, que es la de si tiene un bloque o no. La única casilla meta con la altura distinta respecto al estado inicial aparece siempre en los patrones con más de una variable. Emplea poca memoria. RESUELTO.

- Problema 2: Un patrón *grande* (de aproximadamente 10 variables) con la variable 0 (la posición del robot), las metas con las *alturas variables* y el *hasblock*. No aparecen las variables 2, 3, 4 ni 6, los cuales pertenecen a posiciones por los que el robot pasa y

construye y otros no. El patrón mediano tiene el 0, uno de altura variable y uno por el que pasa. Los pequeños no tienen las variables que aparecen en el patrón *mediano*. Se emplea poca memoria. Solo hay dos metas distintas al *init*. RESUELTO.

- Problema 3: Los patrones grandes (solo 3 y de aproximadamente 10 variables) tienen la posición del robot y aparecen casi todas las variables. Las PDBs individuales tienen todas excepto el 12, que es la de si tiene un bloque o no. La casilla meta con la altura distinta aparece siempre en los patrones con más de 1 variable. Solo hay una meta distinta al *init*. Se emplea, además, poca memoria. RESUELTO.

- Problema 4: Se forman varios conjuntos aditivos. Sin embargo, el patrón más largo tiene las *alturas variables* (2). También aparece el 4 con el 0 y el 5 con el 0, siendo el 4 y el 5 *alturas variables*. Cuando un patrón *mediano* tiene unas variables, estas variables no aparecen como patrones individuales. Se emplea, además, poca memoria. RESUELTO.

- Problema 5: Igual que en los problemas 1 y 3, pero con 4 patrones largos. RESUELTO.

- Problema 6: semejante al problema 2, pero con otras variables que no aparecen en el grande.

- Problema 7: Igual que el problema 2, pero con una meta distinta al *init* (aunque grande, de altura 6).

- Problema 8: igual que el problema 7.

- Problema 9: igual que el problema 1, pero llega a altura 7.

- Problema 10: igual que el problema 1 pero con 2 alturas variables.

- Problema 11: Igual que el problema 2, pero con dos patrones grandes, ambos con todas las alturas variables. El mediano tiene la variable 0 y dos alturas variables. Hay 3 alturas variables en total. RESUELTO.

- Problema 12: Parecido al problema 11, pero con 4 alturas variables y la meta mediana con 4 variables. RESUELTO.

- Problema 13: Nada fuera de lo común que no se haya visto antes. RESUELTO.

- Problema 14: Nada fuera de lo común que no se haya visto antes, pero con 4 metas variables. RESUELTO.

- Problema 15: Nada fuera de lo común que no se haya visto antes.

- Problema 16: Nada fuera de lo común que no se haya visto antes. A partir de aquí hay muchas metas variables.

- Problema 17: Nada fuera de lo común que no se haya visto antes. RESUELTO.

- Problema 18: Caso extraño como el 4 (parecidas características). RESUELTO.

- Problema 19: Nada fuera de lo común que no se haya visto antes. RESUELTO.

- Problema 20: Nada fuera de lo común que no se haya visto antes. RESUELTO.

Llegados hasta aquí, y tras contemplar las PDBs de cada problema, generalizar es complicado, pues el número de variables que hay es muy pequeño, y muchas de ellas son variables meta. Por esa razón, además de observar con más detalle aquellos problemas resueltos, se utilizará la lógica y el sentido común para implementar el algoritmo.

En primer lugar, no se podrían incorporar todas las variables en una PDB en todos los problemas, porque, si se pudiese hacer, el algoritmo *combo* mostraría mejores resultados de los que muestra. Es por eso por lo que es necesario saber elegir bastante bien qué variables relacionadas con las alturas incorporar. Dicho esto, la mejor alternativa es construir tres tipos de PDBs:

- Una PDB con la variable que indica la posición del robot y cada una de las variables que hagan referencia a casillas en las que la meta y el estado inicial tengan distintas alturas. A estas variables se les llamarán, a partir de ahora, alturas variables.

- Un conjunto de PDBs resultado de todas las combinaciones entre todas las casillas adyacentes (excluyendo la cantera) a las alturas variables, añadiéndole también a cada una de ellas el *hasblock* y la posición del robot. Por ejemplo, si un problema tuviese dos alturas variables (*H1* y *H2*), y una de ellas tuviese 3 vecinos (*A, B* y *C*) y la otra solo dos (*X* y *Z*), (no siendo ninguna de ellas la posición de la cantera) el número de PDBs sería de 6: [*H1*, *H2*, *A*, *X*, *hasblock* y la posición del robot], [*H1*, *H2*, *A*, *Z*, *hasblock* y la posición del robot], [*H1*, *H2*, *B*, *X*, *hasblock* y la posición del robot], [*H1*, *H2*, *B*, *Z*, *hasblock* y la posición del robot], [*H1*, *H2*, *C*, *X*, *hasblock* y la posición del robot], [*H1*, *H2*, *C*, *Z*, *hasblock* y la posición del robot]. Esto tiene mucho sentido puesto que, para que el robot construya en una posición, es necesario visitar alguna de sus casillas adyacentes. Como resulta evidente, todas las variables repetidas se suprimen. Además, las PDBs superiores a 10 variables se descartarán, para que el grafo asociado quepa en memoria y porque es el número de variables que aparece aproximadamente en la resolución de los problemas.

Considerar además los vecinos de los vecinos de las alturas variables daba como lugar una gran explosión combinatoria innecesaria.

- Una PDB *larga* con todas las variables que han aparecido en las PDBs nombradas anteriormente, siempre y cuando el número de variables no superase un determinado valor, para que no diese error. Ese número se fijó en 10, pues es el número máximo de variables que han aparecido en una PDB.

Además, se pondrán todas las variables que tienen que ver con la altura en una PDB individual, excepto las que representen alturas variables, pues aparecen en todas las PDBs. Esto fuerza a que todas las variables aparezcan en, al menos, una PDB. Primero se colocan los patrones más grandes, y luego los patrones individuales, con el fin de favorecer a *Zero-One PDB*, aunque para *Canonical PDB* no sea relevante la ordenación.

4.3.3 Implementación y ejecución del algoritmo

En este punto se implementará el algoritmo de acorde a la generalización realizada en el anterior punto. El *script* implementado se comenta en el punto 3.3.4 *Script* de algoritmo para *Termes*.

Los resultados, dado que se van a comparar posteriormente con el resto de los algoritmos, se muestran en conjunto (*tablas 138 – 140 y 197 – 200*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgTermes (Canonical PDB)*	*AlgTermes (Zero-One PDB)*
P01	3.60109	0.698697	0.671691	0.11834303	0.116089
P02	3.58253	3.78121	3.82217	0.62873187	0.580669
P03	4.85755	17.0263	16.9636	6.09257109	6.27562
P04	7.34706	34.8623	34.379	5.56850395	5.60332
P05	271.911	-	-	-	-
P06	-	-	-	-	-
P07	-	-	-	-	-
P08	-	-	-	-	-

P09	-	-	-	-	-
P10	-	-	-	-	-
P11	5.7526	11.0482	10.788	1.23375510	1.30926
P12	3.60016	2.2896	2.27298	1.28281204	1.49435
P13	58.1779	78.4104	77.4558	53.44804313	76.904
P14	11.7071	236.067	229.375	35.62901007	37.3906
P15	-	-	-	-	-
P16	-	-	-	-	-
P17	81.8209	188.883	145.432	100.35224798	143.41
P18	15.6232	46.6441	32.7036	14.63828609	24.472
P19	27.0189	91.109	74.7636	58.01016885	79.1927
P20	126.88	328.406	243.166	73.85213813	121.824

Tabla 138. Tiempo total – Termes – Evaluación

Número nodos expandidos	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTermes (Canonical PDB)***	***AlgTermes (Zero-One PDB)***
P01	4776	47277	47286	25356	25972
P02	17874	995688	995848	153247	153435
P03	145662	4980609	4989089	1630019	1646631
P04	748085	10104292	10105099	1449714	1449986
P05	74493127	-	-	-	-

P06	-	-	-	-	-
P07	-	-	-	-	-
P08	-	-	-	-	-
P09	-	-	-	-	-
P10	-	-	-	-	-
P11	219376	3105725	3105766	137102	137116
P12	52374	533081	533241	71225	91595
P13	14958320	23667345	23672375	13849206	20185591
P14	1953682	69375311	69387188	9693366	9693556
P15	-	-	-	-	-
P16	-	-	-	-	-
P17	22443585	45661696	45661896	22629895	32242343
P18	3403414	9807649	9807890	2316184	4676981
P19	6027955	22591891	22591922	3832006	7621380
P20	34655820	74591971	74592951	17442585	31815333

Tabla 139. Número nodos expandidos – Termes – Evaluación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgTermes (Canonical PDB)*	*AlgTermes (Zero-One PDB)*
P01	36	36	36	36	36
P02	54	54	54	54	54

P03	68	68	68	68	68
P04	80	80	80	80	80
P05	132	-	-	-	-
P06	-	-	-	-	-
P07	-	-	-	-	-
P08	-	-	-	-	-
P09	-	-	-	-	-
P10	-	-	-	-	-
P11	66	66	66	66	66
P12	46	46	46	46	46
P13	92	92	92	92	92
P14	104	104	104	104	104
P15	-	-	-	-	-
P16	-	-	-	-	-
P17	116	116	116	116	116
P18	76	76	76	76	76
P19	94	94	94	94	94
P20	106	106	106	106	106

Tabla 140. Coste de la solución – Termes – Evaluación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.3.4 Pruebas del algoritmo

Estas pruebas consisten en seleccionar 3 problemas. Es favorable que ofrezcan resultados diferentes entre sí, para probar mejor el algoritmo. Se compara la salida esperada con la obtenida, y si coinciden, se podría verificar su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validaría.

Se van a observar los problemas 1, 10 y 12. En el primero de ellos solo hay una altura variable, en el segundo hay dos y comparten vecinos entre sí, y en el tercero hay 4 con vecinos en común y vecinos que coinciden con la cantera.

- El algoritmo implementado sobre el problema 1 muestra las siguientes PDBs:

[0, 4, 8, 9, 11, 12]

[0, 4, 8, 12]

[0, 8, 9, 12]

[0, 8, 11, 12]

[0, 8]

[11]

[10]

[9]

[7]

[6]

[5]

[4]

[3]

[2]

[1]

0: la posición del robot.

12: *hasblock*.

1 – 11: posiciones del tablero.

8: *altura variable*.

- El resto de las pruebas se encuentran en la Sección 9 Anexo.

Tal y como se puede observar, aparece una PDB individual por cada variable relacionada con la altura de una casilla que no sea una *altura variable*. Aparece otra PDB con la posición del robot y las alturas variables. El resto de PDBs tienen la posición del robot, todas las alturas variables, el *hasblock* y una combinación de vecinos. A continuación, se demuestra que es correcto:

- Problema 1: la única variable meta tiene 3 vecinos (2, 9 y 11), con lo cual aparecen 3 PDBs.

- Problema 10: aparecen dos variables meta (4 y 6). El 6 tiene tres vecinos (2, 4 y 11), y el 4 tiene dos (6 y 8). Además, el 4 y el 6 son vecinos entre sí. Por lo que el número de combinaciones es 6, suprimiendo aquellas variables repetidas.

- Problema 12: aparecen cuatro variables meta (1, 4, 6 y 11). El 1 tiene dos vecinos (la cantera y el 6), el 4 tiene tres (la cantera, el 3 y el 11), el 6 tiene tres (el 1, el 2 y el 10), y el 11 tiene cuatro (el 4, el 7, el 9 y el 10). Realizando la combinatoria y suprimiendo las alturas variables, la cantera y las variables repetidas, se obtiene el siguiente conjunto, que coincide con las PDBs mostradas anteriormente: [-], [2], [3], [7], [9], [10], [2, 3], [2, 7], [2, 9], [2, 10], [3, 7], [3, 9], [3, 10], [7, 10], [9, 10], [2, 3, 7], [2, 3, 9], [2, 3, 10], [3, 7, 10] y [3, 9, 10]

Para finalizar, se observa otra PDB más con todas las variables que aparecen en esta combinatoria. Ninguna de las PDB tiene más de 10 variables. Y, como resulta evidente, todas las PDBs se encuentran ordenadas de mayor a menor tamaño.

Dicho esto, el algoritmo funciona correctamente. Queda, por tanto, verificado y validado, pues lo importante del algoritmo, dado que es sencillo, son las PDBs que construye. Dadas las características del *Script*, un análisis más exhaustivo relacionado con una metodología más dura no tendría sentido.

4.3.5 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con el resto de los algoritmos.

En primer lugar, respecto al tiempo de generación de las PDBs, el algoritmo implementado no requiere de mucho tiempo, pero llegando a superar a *iPDB* cuando el número de alturas variables es muy alto (aumentando así el número de combinaciones y de PDBs). En el peor de los casos no llega a los 43 segundos. Tanto en *Zero-One* como en *Canonical* los resultados son parecidos. Esto no se debe al algoritmo de generación de PDBs en sí, sino a la construcción de los grafos y tablas necesarias relacionadas con

esos patrones. Por otra parte, los tiempos de poda y obtención de los conjuntos aditivos es despreciable.

Respecto al tiempo de búsqueda, *iPDB* es la mejor alternativa cuando los problemas tienen pocas alturas variables, pues este genera pocas PDBs pero con muchas variables, al contrario que *AlgTermes*. Pero en caso contrario, en algunas ocasiones la mejor elección es *AlgTermes* en *Canonical*, pues se centra más en la combinatoria entre los vecinos, aunque a veces *iPDB* lo supera. Por otra parte, el uso de *Zero-One* no es tan aconsejado si lo que se desea es reducir el tiempo de búsqueda.

Dicho esto, los algoritmos que menos tiempo requieren son el recién implementado e *iPDB*. Elegir uno entre estos dos es complicado, pues depende del problema en sí. Se observa que en problemas sencillos *AlgTermes* mediante *Canonical* es el que menos tiempo global consume. En problemas de dificultad media, *AlgTermes* suele proporcionar mejores resultados en general, pero a veces iPDB consigue tiempos más pequeños. En problemas con muchas alturas variables en ocasiones *iPDB* obtiene mejores marcas y otras veces no. Ni los algoritmos *combo* ni *AlgTermes* en *Zero-One* pueden competir contra las marcas temporales de estos dos algoritmos.

Respecto a la memoria utilizada, hay bastantes cosas para tener en cuenta. La primera es que, cuando *iPDB* construye patrones con muchas variables en problemas sencillos, el consumo de memoria se dispara. Sin embargo, esos mismos patrones largos en problemas un poco más complejos muestra un consumo de memoria más pequeño que en cualquiera de los otros cuatro algoritmos. Por otra parte, en problemas de dificultad intermedia, *AlgTermes* en *Canonical* suele ser el que menos memoria consume. En los problemas con muchas alturas variables en ocasiones *iPDB* consume más memoria que el algoritmo implementado (en *Canonical*) otras veces no. Ni los algoritmos *combo* ni *AlgTermes* en *Zero-One* pueden competir con estos algoritmos en cuestión de consumo de memoria, a excepción de los problemas sencillos, en los que *Zero-One* no dista mucho de *Canonical*.

Respecto al número de nodos expandidos, dato que refleja la calidad de la heurística, *iPDB* supera en calidad al implementado mediante *Canonical* en problemas sencillos, pues crea PDBs con casi todas las variables. En problemas de dificultad media, a veces gana *iPDB* y a veces *AlgTermes*, dependiendo si hay o no muchas alturas variables. Sin embargo, cuando el número de estas alturas variables aumenta, *AlgTermes* expande menos nodos que *iPDB*. El resto de los algoritmos no pueden competir contra estos dos en *Termes*.

Por último, el coste de la solución de todos los algoritmos es el mismo, como cabría esperar. Sin embargo, *iPDB* resuelve un problema más que el resto, asociado a un tablero con pocas alturas variables. *iPDB* suele dar mejores resultados en problemas con pocas alturas variables.

4.3.6 Extracción de conclusiones

Tras observar las anteriores comparaciones es difícil establecer unas conclusiones claras acerca de cuál es el mejor algoritmo. En este apartado tan solo se realizarán aquellas más generales.

En primer lugar, lo que está claro es que ni los algoritmos *combo* ni *AlgTermes* en Z*ero-One* son capaces de competir contra el algoritmo implementado en *Canonical* ni contra *iPDB*. Luego, respecto a la memoria utilizada, los dos algoritmos restantes compiten muy equilibradamente. Respecto al tiempo, *AlgTermes* en *Canonical* proporciona tiempos mejores con mayor frecuencia, incluyendo los problemas más complejos. Por otra parte, como resulta evidente, la heurística de *iPDB* cuando incorpora casi todas las variables en una PDB es más informada, pero debido al tiempo de construcción de estas, no merece tanto la pena. En problemas en los que el número de alturas variables es pequeño, parece indicar que *iPDB* también lleva la delantera, pues es capaz de resolver un problema más que el resto, a pesar de que esto le suponga un consumo más elevado de tiempo. Sin embargo, en problemas con más alturas variables, *AlgTermes* en *Canonical* destaca.

iPDB es apropiado, por tanto, cuando se desee resolver un problema con pocas metas variables (su búsqueda mediante *Hill climbing* suele ser muy apropiada para elegir las variables que mejor pueden funcionar), y *AlgTermes* en caso contrario (debido en mayor parte por la combinatoria de casillas vecinas). Sin embargo, dado que *AlgTermes* suele proporcionar tiempos menores, es aconsejable ejecutar siempre en este algoritmo tras ejecutar en *iPDB*, con el fin de mejorar la marca temporal (aunque no es seguro que pueda resolverlo). De igual forma, es posible conseguir tiempos mejores si se ejecuta en *iPDB* tras ejecutar en *AlgTermes*. Elegir uno de los dos algoritmos es difícil, pues depende del problema en sí; pero si tuviese que elegir uno me decantaría por *AlgTermes* mediante *Canonical*, pues *iPDB* no destaca sobre este en ninguna de las anteriores tablas (exceptuando la del número de problemas resueltos). El número de problemas resueltos de más por *iPDB* respecto a *AlgTermes* es tan solo de 1, y se corresponde con un problema con tan solo una *altura variables*, es decir, un tipo de problema en el que *iPDB* destaca. No se debe a que *iPDB* sea mejor. *AlgTermes* en *Canonical* ha proporcionado, por lo general, mejores resultados.

4.4 Dominio *Hiking*

4.4.1 Ejecución de los algoritmos

El primer paso es ejecutar los algoritmos seleccionados en el punto 4.1.2 Selección de algoritmos. Los resultados son los siguientes (*tablas 141 – 143 y 201 – 204*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	0.032566	0.00313647	0.0104389
P02	0.0783462	0.0122534	0.0108647
P03	0.127391	0.0154942	0.0249672
P04	0.272573	0.0942165	0.0520124
P05	0.518506	0.140937	0.145938
P06	0.209041	0.328808	0.315975
P07	3.69358	2.73616	2.41973
P08	52.8204	37.0871	35.3655
P09	450.684	395.504	381.722
P10	-	-	-
P11	-	-	-
P12	14.7052	3.71146	3.67457
P13	269.414	225.357	216.139
P14	-	-	-
P15	-	-	-
P16	0.877289	1.07354	0.989832
P17	48.6175	9.47674	9.30874
P18	-	172.16	166.076

P19	-	-	-
P20	-	-	-

Tabla 141. Tiempo total – Hiking – Observación

Número nodos expandidos	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	26	12	12
P02	301	18	18
P03	2140	26	26
P04	30812	39	39
P05	77262	46	46
P06	10398	11	11
P07	477813	260215	260215
P08	6901037	4324239	4324239
P09	59226603	48258370	48258370
P10	-	-	-
P11	-	-	-
P12	1548040	262505	262505
P13	28004230	21658450	21658450
P14	-	-	-
P15	-	-	-

P16	48402	9858	9858
P17	4107620	579825	579825
P18	-	10861732	10861732
P19	-	-	-
P20	-	-	-

Tabla 142. Número nodos expandidos – Hiking – Observación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	11	11	11
P02	17	17	17
P03	25	25	25
P04	38	38	38
P05	45	45	45
P06	10	10	10
P07	16	16	16
P08	22	22	22
P09	30	30	30
P10	-	-	-
P11	-	-	-
P12	17	17	17

P13	24	24	24
P14	-	-	-
P15	-	-	-
P16	10	10	10
P17	17	17	17
P18	-	23	23
P19	-	-	-
P20	-	-	-

Tabla 143. Coste de la solución – Hiking – Observación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.4.2 Observación de resultados

En primer lugar, se justifican los motivos por los cuales no se han introducido ciertos parámetros:

- ***pdb_max_size***: a pesar de que las PDBs contienen pocas variables, el número total de variables en los *output* es también pequeño. Además, es razonable que, en problemas con pocas variables meta, al final se formen pocas PDBs y con pocas variables. Aumentar este valor, dadas las condiciones del dominio y de los problemas, no tiene sentido.

- ***collection_max_size***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente dado el poco número de variables totales, de los cuales, pocas son las relacionadas con metas. De todos modos, el algoritmo de creación de PDBs interrumpía su proceso debido a que ninguna nueva PDB superaba el *min_improvement*.

- ***min_improvement***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente para hacer un buen análisis.

- ***max_time*** el tiempo de generación de las PDBs es, de por sí, muy pequeño.

- ***max_time_dominance_pruning*** **(en *iPDB*)**: el tiempo de poda es muy pequeño.

- ***max_time_dominance_pruning*** **(en *Canonical PDB*)**: el tiempo de poda es muy pequeño.

En primer lugar, en ninguno de los algoritmos el tiempo de generación de las PDBs es significativo. Todos se sitúan debajo del segundo y medio. Igualmente, el tiempo de poda y generación de los conjuntos aditivos es igualmente despreciable. Sin embargo, respecto al tiempo de búsqueda, ambos algoritmos *combo* obtienen mejores marcas. Estos algoritmos *combo* son los que menos tiempo total consume.

Respecto a la cantidad de memoria consumida, en los problemas sencillos *iPDB* consume un poco menos de memoria que los otros dos algoritmos, debido a la longitud de las PDBs generadas por los *combo*. Sin embargo, en problemas más complejos, *iPDB* termina utilizando más recursos de almacenamiento. Entre los dos algoritmos *combo*, no existen diferencias significativas respecto a la memoria. Con el número de nodos expandidos pasa lo mismo, iPDB no es capaz de competir contra los *combo*, y entre ellos no existe ninguna diferencia.

Además, los algoritmos *combo* son capaces de resolver un problema más que *iPDB*. Y, como resulta evidente, los costes de los planes generados son exactamente los mismos. Esto no implica que los planes sean igual, pues puede darse el caso de que exista más de un plan meta con el mismo coste.

Se puede llegar a la conclusión de que, mediante *Canonical PDB*, la heurística es igual de informada que en *Zero-One* usando *combo*, debido a que el tamaño del conjunto de patrones es de 1. Dicho esto, no merece la pena emplear tiempo de más en poda, construcción de conjuntos aditivos y consulta a varias tablas, cuando solo ha construido una PDB.

Sin embargo, realizar un estudio de los patrones que el algoritmo *combo* aporta en este dominio tan pequeño no es la mejor alternativa. Se puede llegar a la conclusión de que los algoritmos *combo* ejecutados no utilizan todas las variables en todos los problemas, pues, de ser así, el tiempo de ejecución del planificador no llegaría a un segundo; y el número de nodos expandidos sería igual prácticamente el mismo que el coste del plan solución, ya que la heurística sería perfecta. Dadas las características del dominio y del problema sería una mejor idea realizar un estudio de *iPDB*, pero teniendo en mente que, cuantas más variables estén dentro de un mismo patrón, mejor tenderá a funcionar el algoritmo, debido a los buenos resultados del algoritmo *combo*. Podrían encontrarse patrones de comportamiento interesantes si se analizasen las PDBs en cuestión. Si solo se realizase el estudio con este mejor algoritmo, no se sabría muy bien qué variables en concreto incorporar, pues solo se tiene la certeza de que sería acertado introducir una cantidad más o menos extensa de variables en una misma PDB. En resumidas cuentas, el estudio en *iPDB* mostrará cuáles son las variables relevantes, y serán estas las que se

introducirán en una PDB, teniendo constancia de que en un mismo patrón se puede incorporar un número relativamente grande de variables.

Las características observadas de cada problema donde el algoritmo iPDB ha actuado, realizando previamente la poda y la creación de conjuntos aditivos, son las siguientes:

- Problema 1: una PDB con el estado de la tienda de campaña, las posiciones de la tienda de campaña y de las personas y el *walked* de la pareja. RESUELTO.

- Problema 2: Igual que el 1. RESUELTO.

- Problema 3: Igual que el 1. RESUELTO.

- Problema 4: Igual que el 1. RESUELTO.

- Problema 5: Igual que el 1. RESUELTO.

- Problema 6: Dos PDBs con la posición de cada miembro de la pareja y su *walked* (3 variables) y dos PDBs individuales con los *walked*. RESUELTO.

- Problema 7: Igual que el 6. RESUELTO.

- Problema 8: Igual que el 6. RESUELTO.

- Problema 9: Igual que el 6. RESUELTO.

- Problema 10: Igual que el 6.

- Problema 11: Igual que el 6.

- Problema 12: Igual que el 6. RESUELTO.

- Problema 13: Igual que el 6. RESUELTO.

- Problema 14: Igual que el 6.

- Problema 15: Igual que el 6.

- Problema 16: Igual que el 6. RESUELTO.

- Problema 17: Igual que el 6. RESUELTO.

- Problema 18: Igual que el 6.

- Problema 19: Igual que el 6.

- Problema 20: Igual que el 6.

Puesto que es recomendable incluir múltiples variables en una sola PDB (por los buenos resultados del algoritmo *combo*), lo más lógico es crear PDBs en función de las observaciones de los primeros problemas, es decir, tener en cuenta la tienda de campaña. Es por eso por lo que, recopilando toda esta información, podemos obtener este conjunto de PDBs:

- Un patrón individual por cada variable *walked*.

- Un patrón largo por cada *walked* que tiene el problema, junto con cada miembro de la pareja relacionada con dicho *walked*, y todas las variables relacionadas con las tiendas de campaña, es decir, su posición y si se encuentra montada o desmontada.

Primero se colocan los patrones más grandes, y luego los patrones individuales, con el fin de favorecer a *Zero-One PDB*, aunque para *Canonical PDB* no sea relevante la ordenación. Las metas corresponden con los *walked*, por lo que siempre aparecen en el conjunto de PDBs creadas.

4.4.3 Implementación y ejecución del algoritmo

En este punto se implementará el algoritmo de acorde a la generalización realizada en el anterior punto. El *script* implementado se comenta en el punto 3.3.5 *Script* de algoritmo para *Hiking*.

Los resultados, dado que se van a comparar posteriormente con el resto de los algoritmos, se muestran en conjunto (*tablas 144 – 146 y 201 – 204*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgHiking (Canonical PDB)*	*AlgHiking (Zero-One PDB)*
P01	0.032566	0.00313647	0.0104389	0.00318769	0.00301804
P02	0.0783462	0.0122534	0.0108647	0.00661312	0.00653135
P03	0.127391	0.0154942	0.0249672	0.03501052	0.03495632
P04	0.272573	0.0942165	0.0520124	0.18526599	0.24324699
P05	0.518506	0.140937	0.145938	0.36832912	0.42281512

P06	0.209041	0.328808	0.315975	0.03768729	0.06238659
P07	3.69358	2.73616	2.41973	0.19507193	0.55056593
P08	52.8204	37.0871	35.3655	1.49180899	6.42595899
P09	450.684	395.504	381.722	11.31199909	60.34009909
P10	-	-	-	53.02414192	334.81044192
P11	-	-	-	243.73920004	-
P12	14.7052	3.71146	3.67457	0.59621516	1.59029516
P13	269.414	225.357	216.139	6.17371200	22.39603200
P14	-	-	-	42.05324300	187.26624300
P15	-	-	-	274.48803722	-
P16	0.877289	1.07354	0.989832	0.06554133	0.12336503
P17	48.6175	9.47674	9.30874	1.23725816	3.23605816
P18	-	172.16	166.076	14.91013789	47.22063789
P19	-	-	-	146.56743888	-
P20	-	-	-	-	0.05224800

Tabla 144. Tiempo total – Hiking – Evaluación

Número nodos expandidos	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgHiking (Canonical PDB)***	***AlgHiking (Zero-One PDB)***
P01	26	12	12	26	26
P02	301	18	18	301	301

P03	2140	26	26	2140	2140
P04	30812	39	39	30812	30812
P05	77262	46	46	77262	77262
P06	10398	11	11	495	1482
P07	477813	260215	260215	9903	38216
P08	6901037	4324239	4324239	97234	533763
P09	59226603	48258370	48258370	875690	5728706
P10	-	-	-	4335060	32610303
P11	-	-	-	22704671	-
P12	1548040	262505	262505	25862	98859
P13	28004230	21658450	21658450	318545	1577253
P14	-	-	-	2432003	13454525
P15	-	-	-	17011918	-
P16	48402	9858	9858	2119	5560
P17	4107620	579825	579825	46955	178523
P18	-	10861732	10861732	634659	2812027
P19	-	-	-	6751727	-
P20	-	-	-	-	-

Tabla 145. Número nodos expandidos – Hiking – Evaluación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgHiking (Canonical PDB)*	*AlgHiking (Zero-One PDB)*
P01	11	11	11	11	11
P02	17	17	17	17	17
P03	25	25	25	25	25
P04	38	38	38	38	38
P05	45	45	45	45	45
P06	10	10	10	10	10
P07	16	16	16	16	16
P08	22	22	22	22	22
P09	30	30	30	30	30
P10	-	-	-	34	34
P11	-	-	-	42	-
P12	17	17	17	17	17
P13	24	24	24	24	24
P14	-	-	-	28	28
P15	-	-	-	35	-
P16	10	10	10	10	10
P17	17	17	17	17	17

P18	-	23	23	23	23
P19	-	-	-	30	-
P20	-	-	-	-	-

Tabla 146. Coste de la solución – Hiking – Evaluación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.4.4 Pruebas del algoritmo

Estas pruebas consisten en seleccionar 3 problemas. Es favorable que ofrezcan resultados diferentes entre sí, para probar mejor el algoritmo. Se compara la salida esperada con la obtenida, y si coinciden, se podría verificar su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validaría.

Elegir tres problemas distintos es complicado, pues todos son muy similares entre sí. Finalmente, se ha decidido que se van a observar los problemas 1, 7 y 20. Es decir, un problema con una sola pareja y una única tienda de campaña; otro con dos parejas, dos tiendas de campaña y pocos lugares y coches; y el último con dos parejas, dos tiendas de campaña, pero con bastantes lugares y coches.

- El algoritmo implementado sobre el problema 1 muestra las siguientes PDBs:

[0, 3, 4, 5, 6]

[6]

0 y 3: variables relacionadas con la tienda de campaña.

4 y 5: variables relacionadas con la posición de las parejas.

6: variable *walked*.

- El resto de las pruebas se encuentran en la Sección 9 Anexo.

Tal y como se puede observar, aparece una PDB individual por cada variable *walked* del problema. Además, se puede vislumbrar cómo se ha creado una PDB por cada variable *walked*, junto con los miembros de la pareja asociados a dicho *walked* y toda la información acerca de todas las tiendas de campaña. Y, como resulta evidente, todas las PDBs se encuentran ordenadas de mayor a menor tamaño.

Como se puede comprobar, el algoritmo funciona correctamente. Queda, por tanto, verificado y validado, pues lo importante del algoritmo, dado que es sencillo, son las PDBs que construye. Dadas las características del *Script*, un análisis más exhaustivo relacionado con una metodología más dura no tendría sentido.

4.4.5 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con el resto de los algoritmos.

En primer lugar, respecto al tiempo de generación de las PDBs, ninguno de los algoritmos requiere de un tiempo relevante. Igualmente, el tiempo de poda es despreciable. Sin embargo, respecto al tiempo de búsqueda, *AlgHiking* proporciona mejores resultados que el resto con muchísima diferencia. Además, cabe destacar que en *Canonical* los resultados son aún mejores.

Respecto a la memoria utilizada, en problemas en los que solo hay una variable *walked* es difícil establecer qué algoritmo gasta menos memoria. Sin embargo, en problemas más complejos se puede observar, de nuevo, cómo *AlgHiking* utiliza menos recursos de capacidad que el resto, sobre todo cuando se ejecuta sobre *Canonical*.

Observando el número de nodos expandidos se puede llegar a apreciar la calidad de la heurística. En problemas sencillos el algoritmo *combo* ofrece una mejor heurística, pero en estos problemas el tiempo de ejecución total no alcanza el medio segundo. En el resto de los problemas domina el algoritmo implementado y, de nuevo, mejor es la calidad de la heurística si se ejecuta mediante *Canonical*.

Y, por último, el algoritmo implementado mediante *Zero-One* resuelve dos problemas más que los algoritmos *combo*, pero *AlgHiking* con *Canonical* es capaz incluso de resolver otros tres problemas más, alcanzando los 19 problemas resueltos. El coste de los planes es, obviamente, el mismo.

4.4.6 Extracción de conclusiones

Tras observar las anteriores comparaciones se pueden extraer conclusiones respecto al dominio *Hiking*. En este apartado tan solo se realizarán aquellas más generales.

Es fácil llegar a la conclusión de que el estudio de las variables llevado a cabo ha dado unos muy buenos resultados, pues ha originado un algoritmo potente. *AlgHiking* en

Canonical es el mejor de los 5 algoritmos, tal y como se ha podido ver en el anterior punto. El estudio y las decisiones llevadas a cabo han sido un éxito.

4.5 Dominio *Spider*

4.5.1 Ejecución de los algoritmos

El primer paso es ejecutar los algoritmos seleccionados en el punto 4.1.2 Selección de algoritmos. Los resultados son los siguientes (*tablas 147 – 149 y 205 – 208*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	3.89944	0.943754	1.02294
P02	8.88602	0.858228	0.80472
P03	16.5604	3.144	29.4693
P04	45.8777	2.6882	43.1336
P05	52.7484	9.94529	-
P06	-	-	-
P07	3.16054	1.06021	0.904593
P08	4.57249	0.934816	1.23525
P09	14.9599	3.2036	23.8526
P10	132.884	5.63498	202.149
P11	291.744	-	-
P12	-	-	-
P13	919.798	-	-

P14	30.1373	0.717136	0.831224
P15	5.04144	0.694479	0.724024
P16	15.4625	9.37359	131.358
P17	28.105	49.1594	-
P18	427.141	-	-
P19	-	-	-
P20	-	-	-

Tabla 147. Tiempo total – Spider – Observación

Número nodos expandidos	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	2694	6965	8186
P02	4854	6437	7855
P03	32275	89152	2000524
P04	29191	130197	2679528
P05	288174	397845	-
P06	-	-	-
P07	2578	3496	3715
P08	6578	21315	47539
P09	47373	110971	1705033
P10	23897	298678	12161714

P11	11896317	-	-
P12	-	-	-
P13	15093157	-	-
P14	892	8130	13248
P15	561	904	1535
P16	137211	554004	10132376
P17	612546	3164316	-
P18	17748489	-	-
P19	-	-	-
P20	-	-	-

Tabla 148. Número nodos expandidos – Spider – Observación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	16	16	16
P02	23	23	23
P03	22	22	22
P04	31	31	31
P05	50	50	-
P06	-	-	-
P07	16	16	16

P08	18	18	18
P09	25	25	25
P10	27	27	27
P11	29	-	-
P12	-	-	-
P13	39	-	-
P14	10	10	10
P15	17	17	17
P16	17	17	17
P17	23	23	-
P18	27	-	-
P19	-	-	-
P20	-	-	-

Tabla 149. Coste de la solución – Spider – Observación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.5.2 Observación de resultados

En primer lugar, se justifican los motivos por los cuales no se han introducido ciertos parámetros:

- ***pdb_max_size***: variar este parámetro no supondría una mejora, ya que ninguna PDB ha sido descartada por superar el tamaño máximo en ningún problema.

- ***collection_max_size***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente para extraer conclusiones. De todos modos, el algoritmo de creación de PDBs interrumpía su proceso debido a que ninguna nueva PDB superaba el *min_improvement*.

- ***min_improvement***: no es necesario incrementar el tamaño de la colección, ya que el que se obtiene es suficiente para hacer un buen análisis. De todos modos, el algoritmo de generación de PDBs interrumpía su proceso puesto que no se obtenía una nueva PDB con un *improvement* positivo.

- ***max_time***: en todos los problemas se ha podido construir una colección de PDBs a tiempo, sin detener la ejecución en el proceso de construcción de patrones.

- ***max_time_dominance_pruning*** **(en *iPDB*)**: el tiempo de poda es muy pequeño.

- ***max_time_dominance_pruning*** **(en *Canonical PDB*)**: el tiempo de poda es muy pequeño.

En *iPDB*, el tiempo de generación de las PDBs es muy grande respecto a los otros dos algoritmos. Pero el tiempo de búsqueda es mucho menor, al igual que el número de nodos expandidos, pues la heurística que se crea en *iPDB* es más informada. Por otra parte, el tiempo de poda es despreciable en *iPDB* y en *Canonical PDB*. Al igual que en el dominio *Snake*, el tiempo que *iPDB* ahorra en el proceso de búsqueda no llega a compensar el tiempo empleado en generar las PDBs. Pero esta vez, *iPDB* no emplea demasiados recursos de almacenamiento en construir los patrones, pues estos están formados por pocas variables, a pesar de que el número de PDBs sea grande. Los otros dos algoritmos requieren más memoria.

Respecto a los algoritmos *combo*, en *Canonical PDB* y en *Zero-One PDB* las diferencias no son notables. No supera los 3 segundos en ningún problema. Además, otra diferencia con el dominio *Snake* es la gran diferencia en el tiempo de búsqueda entre el algoritmo *combo* de *Canonical*. Dado que el número de PDBs en *combo* es grande (ya que forma bastantes conjuntos aditivos) y estas tienen pocas variables, funciona mejor una interpretación de la heurística como la que se realiza en *Canonical*, por lo que el número de nodos expandidos es mucho menor. Esto queda reflejado en una diferencia de tiempo total y de coste de memoria (pues su gasto en la generación de los conjuntos aditivos se compensa con la memoria gastada en el proceso de búsqueda) a favor del algoritmo *combo* de *Canonical* frente al de *Zero-One*.

Respecto al número de problemas resueltos, el *combo* de *Canonical* resuelve dos problemas más que en *Zero-One*. Sin embargo, *iPDB* termina resolviendo otros tres problemas más que este primero. Y, como resulta evidente, los costes de los planes generados son exactamente los mismos. Esto no implica que los planes sean iguales, pues puede darse el caso de que existan más de un plan meta con el mismo costo.

Tras este análisis, se puede observar cómo *iPDB*, a pesar del gran consumo de tiempo, es la mejor alternativa hasta ahora, ya que resuelve más problemas. Implementar un algoritmo basándose en el estudio de *iPDB* sobre este dominio es más interesante, pues ofrece un conjunto de PDBs productos de un proceso de búsqueda. Podría encontrarse patrones de comportamiento interesantes si se analizasen las PDBs en cuestión.

Las características observadas en cada problema donde el algoritmo *iPDB* ha actuado, realizando previamente la poda, son las siguientes:

- **Problema 1**: Individuales: las metas (*Atom on* Y *Atom clear* de *piles* y *deals*) exceptuando 0-0-0 y 0-1-1 en *atom on*. Medianos: uno con *currently-collecting-deck* y el *on* del 0-2-0; *current-deal* con *currently-dealing* y el *on* del 0-0-0 y el *on* del 0-1-1. RESUELTO.

- **Problema 2**: Individuales: las metas exceptuando *clear deal* 1 y *on* de 0-2-2. Medianos: *currently-collecting-deck* con on 0-0-2; *currently-dealing* con clear deal 1; current-deal con on 0-2-2; *currently-dealing* con *current-deal*, el *clear* deal 1 y el *on* de 0-2-2. RESUELTO.

- **Problema 3**: Individuales: las metas exceptuando on 0-1-1. Medianos: *currently-collecting-deck* con *on* 0-0-3; *current-deal* con *currently-dealing* y *on* 0-1-1; *current-deal* con *currently-dealing*, on 0-1-1 y *clear deal* 0. RESUELTO.

- **Problema 4**: Individuales: metas - 0-0-0, 0-1-0, *clear deal* 1, 0-2-0. 0-3-0. Medianos: *collect-card* 0-0-0 con 0-0-0; *collect-card* 0-1-0 con 0-1-0; *collect-card* 0-2-0 con 0-2-0; *collect-card* 0-3-0 con 0-3-0; *currently-dealing* con *clear* deal 1. RESUELTO.

- **Problema 5**: Individuales: metas - 0-0-3. Medianos: *currently-collecting-deck* con 0-0-4; *current-deal* y *currently-dealing* con 0-0-3; *current-deal* y *currently-dealing* con *clear d*eal 0 y 0-0-3. RESUELTO.

- **Problema 6**: Individuales: metas - *clear deal* 1. Medianos: *currently-collecting-deck* y 0-1-4; *currently-dealing*, *current-deal, clear deal* 1; y *currently-dealing*, *current-deal*, *clear deal* 1 y 0-0-7.

- **Problema 7**: Individuales: metas - 0-0-0 y 0-1-0. Medianos: *currently-collecting-deck* y 1-0-1; *current-deal* y *currently-dealing*, 0-0-0, 0-1-0. RESUELTO.

- **Problema 8**: Individuales: metas - 0-0-0. Medianos: *currently-collecting-deck*, 012; 000, *current-deal* y *currently-dealing*. RESUELTO.

- **Problema 9**: Individuales: metas - 001 y 002. Medianos: *currently-collecting-deck* y 100; *current-deal* y *currently-dealing*, 001 y 002. RESUELTO.

- **Problema 10**: Individuales: metas - 000, 010, 100, 003 y 110. Medianos: *collect-car*d 000 con 000, *collect-card* 010 con 010; 100 con *collect-card* 100; *currently-dealing* con *current-deal* y *collect-card* 003; 110 con *collect-card* 110; *currently-dealing* con *current-deal* y *collect-card* 003 y *clear deal* 0. RESUELTO.

- **Problema 11**: Individuales: metas - 001. Medianos: *currently-collecting-deck* y 100; *current-deal* y *currently-dealing*, *clear deal* 1; *current-deal* y *currently-dealing*, *clear deal* 1, 001. RESUELTO.

- **Problema 12**: Individuales: metas - 001 y 110. Medianos: *currently-collecting-deck* y 016; *current-deal* y *currently-dealing*, 001 y 110.

- **Problema 13**: Individuales: metas - 000, 010, 110, 006, 100. Medianos: 000 y *collect-card* 000; 010 y *collect-card* 010; 110 y *collect-card* 110; 100 y *collect-card* 100; *current-deal* y *currently-dealing* y 006; *clear deal* 0, *current-deal* y *currently-dealing* y 006. RESUELTO.

- **Problema 14**: Individuales: metas - 100, 201. Medianos: *currently-collecting-deck* 100; *current-deal* y *currently-dealing* y 201; *current-deal* y *currently-dealing* y *in-play* 201; 300 y *collect-card* 300; 200 y *collect-card* 200; 100 y *collect-card* 100; 000 y *collect-card* 000. RESUELTO.

- **Problema 15**: Individuales: metas - clear deal 1. Medianos: currently-collecting-deck y 201; *current-deal* y *currently-dealing*, *clear deal* 1; *current-deal* y *currently-dealing*, *clear deal* 1 y 003. RESUELTO.

- **Problema 16**: Individuales: metas - *clear deal* 1. Medianos: cur*rently-collecting-deck*, y 202; *current-deal* y *currently-dealing* y *clear* deal 1; *current-deal* y *currently-dealing*, *clear deal* 1 y 104. RESUELTO.

- **Problema 17**: Individuales: metas - clear deal 1. Medianos: *currently-collecting-deck* y 002; *current-deal* y *currently-dealing* y *clear deal* 1; *current-deal* y *currently-dealing* y *clear deal* 1 y 105. RESUELTO.

- **Problema 18**: Individuales: metas - *clear deal* 1. Medianos: *currently-collecting-deck* y 002; *current-deal* y *currently-dealing* y *clear deal* 1; *current-deal* y *currently-dealing* y *clear deal* 1 y 106. RESUELTO.

- **Problema 19**: Individuales: metas - *clear deal* 1. Medianos: *currently-collecting-deck* y 205; *current-deal* y *currently-dealing* y *clear deal* 1; *current-deal* y *currently-dealing* y clear deal 1 y 107.

- **Problema 20**: Individuales: metas - clear deal 1. Medianos: *currently-collecting-deck* y 005; *current-deal* y currently-dealing y *clear* deal 1; *current-deal* y *currently-dealing* y *clear deal* 1 y 008.

Es necesario indicar que se ha tenido en cuenta el conjunto de PDBs obtenidas tras la poda, pero no los conjuntos aditivos, pues estos se forman mediante *Canonical* directamente. Dicho esto, se ha decido crear las siguientes PDBs:

- Una PDB pequeña por cada una de las metas, pues no se encuentra un patrón claro y sencillo para podar aquellas que el algoritmo *Canonical* terminará desechando. Se observan también ciertas peculiaridades, como que los *clear deal* 1 aparecen cuando hay más de un paquete de cartas (es decir, que existe una carta que se escribe de la siguiente forma: *d1-sx-vy*; siendo *x* e *y* números naturales), pero lo que se quiere hacer es generalizar y aplicar el mismo algoritmo para construir las PDBs siempre de la misma manera. No elegir entre algoritmo dependiendo de las características del problema en cuestión. Esto podría tenerse en cuenta en algún otro trabajo alternativo.

- Como en el 80% de las veces ha salido *currently-collecting-deck* con alguna carta de la *pile*, hacer todas las combinaciones de entre estos dos valores podría ser una buena idea.

- La mayoría de las cartas que aparecen con los *current-deal* y *currently-dealing* son cartas en *deal* y las variables *clear deal*. Dicho esto, una buena alternativa es hacer toda la combinación de *current-deal* y *currently-dealing* con las cartas de *deal* (variables *on*) y los *clear deal* (creando PDBs de 3 variables), y toda la combinación de *current-deal* y *current-dealing* con *clear deal* y cartas de *deal* a la vez (creando PDBs de 4 variables).

De esta forma, entre todos los patrones deberían estar todas las variables *meta*. Además, al igual que se ha hecho en el resto de los dominios, primero se colocan los patrones más grandes, y luego los patrones individuales, con el fin de favorecer a *Zero-One PDB*, aunque para *Canonical PDB* no sea relevante la ordenación.

Para finalizar, es necesario aclarar algunas cuestiones:

- Primero, aunque en alguna ocasión esporádica aparezcan las variables *current-deal* y *currently-dealing* con dos cartas de *current deal*, esto no suele pasar a menudo. Dado que lo que se quiere conseguir es generalizar y que no implemente un algoritmo complejo ni complicado, tomar decisiones en función de esta información no resulta una buena idea. Además, esas cartas no tienen ninguna relación aparente, solo que son de *deal*. La explosión combinatoria puede ser muy grande si se tienen todas en cuenta.

- la variable *in-play* solo aparece en la ejecución de un problema, por lo que tampoco es buena idea tenerla en cuenta.

- No se tienen en cuenta las variables *collect-card* ya que suelen aparecer en las PDBs de los problemas que no tienen *currently-collecting-deck*, y son solo 4 de 20. El 75% de los problemas en lo que existen PDBs con *collect-card* se resuelven en menos de 1 segundo, y la otra se termina también resolviendo, aunque tarda más tiempo.

- Las variables que contienen tanto cartas (*on*) como los *clear pile* no se tienen en cuenta a la hora de realizar la combinatoria.

4.5.3 Implementación y ejecución del algoritmo

En este punto se implementará el algoritmo de acorde a la generalización realizada en el anterior punto. El *script* implementado se comenta en el punto 3.3.6 *Script* de algoritmo para *Spider*.

Los resultados, dado que se van a comparar posteriormente con el resto de los algoritmos, se muestran en conjunto (*tablas 150 – 152 y 205 – 208*):

Tiempo total (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSpider (Canonical PDB)*	*AlgSpider (Zero-One PDB)*
P01	3.89944	0.943754	1.02294	0.18232490	0.13522190
P02	8.88602	0.858228	0.80472	0.31191691	0.24146891
P03	16.5604	3.144	29.4693	2.94548195	1.04915595
P04	45.8777	2.6882	43.1336	9.55503807	2.19571807
P05	52.7484	9.94529	-	21.11523395	7.82037395
P06	-	-	-	-	-
P07	3.16054	1.06021	0.904593	0.15002009	0.20975509
P08	4.57249	0.934816	1.23525	0.37411991	0.33268491
P09	14.9599	3.2036	23.8526	1.77326501	1.28919501

P10	132.884	5.63498	202.149	14.37786917	3.30433917
P11	291.744	-	-	2290.89115784	283.11515784
P12	-	-	-	-	-
P13	919.798	-	-	-	-
P14	30.1373	0.717136	0.831224	0.19013803	0.14392103
P15	5.04144	0.694479	0.724024	0.20347096	0.25848996
P16	15.4625	9.37359	131.358	9.15955011	3.26207011
P17	28.105	49.1594	-	26.27632295	14.22202295
P18	427.141	-	-	-	-
P19	-	-	-	-	-
P20	-	-	-	-	-

Tabla 150. Tiempo total – Spider – Evaluación

Número nodos expandidos	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgSpider (Canonical PDB)***	***AlgSpider (Zero-One PDB)***
P01	2694	6965	8186	2367	2808
P02	4854	6437	7855	4644	4815
P03	32275	89152	2000524	32211	39925
P04	29191	130197	2679528	79811	86492
P05	288174	397845	-	218710	248806
P06	-	-	-	-	-

P07	2578	3496	3715	2455	3040
P08	6578	21315	47539	6538	7936
P09	47373	110971	1705033	42805	51900
P10	23897	298678	12161714	130832	158754
P11	11896317	-	-	11856023	14289345
P12	-	-	-	-	-
P13	15093157	-	-	-	-
P14	892	8130	13248	2765	4371
P15	561	904	1535	577	645
P16	137211	554004	10132376	103061	218488
P17	612546	3164316	-	433257	933749
P18	17748489	-	-	-	-
P19	-	-	-	-	-
P20	-	-	-	-	-

Tabla 151. Número nodos expandidos – Spider – Evaluación

Coste de la solución	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSpider (Canonical PDB)*	*AlgSpider (Zero-One PDB)*
P01	16	16	16	16	16
P02	23	23	23	23	23
P03	22	22	22	22	22

P04	31	31	31	31	31
P05	50	50	-	50	50
P06	-	-	-	-	-
P07	16	16	16	16	16
P08	18	18	18	18	18
P09	25	25	25	25	25
P10	27	27	27	27	27
P11	29	-	-	29	29
P12	-	-	-	-	-
P13	39	-	-	-	-
P14	10	10	10	10	10
P15	17	17	17	17	17
P16	17	17	17	17	17
P17	23	23	-	23	23
P18	27	-	-	-	-
P19	-	-	-	-	-
P20	-	-	-	-	-

Tabla 152. de la solución – Spider – Evaluación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.5.4 Pruebas del algoritmo

Estas pruebas consisten en seleccionar 3 problemas. Es favorable que ofrezcan resultados diferentes entre sí, para probar mejor el algoritmo. Se compara la salida esperada con la obtenida, y si coinciden, se podría verificar su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validaría.

Sin embargo, escoger esos tres problemas no es fácil, pues todos se parecen bastante entre sí. Aquellos que marcan una gran diferencia son los problemas que utilizan una sola baraja y aquellos que usan más de una. Una carta se escribe de la siguiente forma: *dz-sx-vy*; siendo *z*, *x* e y números naturales. *z* toma el valor correspondiente a la baraja en cuestión, *x* es el color del palo e *y* es el valor de la carta. Dicho esto, se van a observar los problemas 1, 10 y 15. Un problema con cartas de una baraja, otro con dos barajas y el último con cuatro.

- El algoritmo implementado sobre el problema 1 muestra las siguientes PDBs:

[0, 1, 156, 161]

[0, 1, 156, 162]

[0, 1, 156, 163]

[0, 1, 156, 164]

[0, 1, 156, 165]

[0, 1, 156, 166]

[0, 1, 157, 161]

[0, 1, 157, 162]

[0, 1, 157, 163]

[0, 1, 157, 164]

[0, 1, 157, 165]

[0, 1, 157, 166]

[0, 1, 156]

[0, 1, 157]

[0, 1, 161]

[0, 1, 162]

[0, 1, 163]

[0, 1, 164]

[0, 1, 165]

[0, 1, 166]

[98, 153]

[98, 154]

[98, 155]

[98, 158]

[98, 159]

[98, 160]

[169]

[168]

[167]

[166]

[165]

[164]

[163]

[162]

[161]

[160]

[159]

[158]

[157]

[156]

[155]

[154]

[153]

0: *current-deal*.

1: *currently-dealing*.

98: *currently-collecting-deck*.

156 y 157: los *clear deal*.

153 – 155 y 158 – 160: cartas de *pile*.

161 – 166: cartas de *deal*.

167 – 169: los *clear* de *pile*.

- El resto de las pruebas se encuentran en la Sección 9 Anexo.

Se puede observar claramente cada conjunto de PDBs creado, y cómo se ordenan de mayor a menor tamaño: las PDBs con las variables meta (los *on* de *pile*, los *on* de *deal*, los *clear* de *pile* y los *clear* de *deal*), una PDB por cada carta de *pile* (con dicha carta y la variable con *currently-collecting-deck*), una PDB por cada carta de *deal* y por cada *clear* de *deal* (con dicha variable y las variables de *current-deal* y *currently-dealing*), y otras PDBs por cada combinación entre las cartas de *deal* y los *clear* de *deal* (con esas dos variables y las variables de *current-deal* y *currently-dealing*).

Como se puede comprobar, el algoritmo funciona correctamente. Queda, por tanto, verificado y validado, pues lo importante del algoritmo, dado que es sencillo, son las PDBs que construye. Dadas las características del *Script*, un análisis más exhaustivo relacionado con una metodología más dura no tendría sentido.

4.5.5 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con el resto de los algoritmos.

En primer lugar, respecto al tiempo de generación de las PDBs, *AlgSpider* requiere poco tiempo. En el peor de los casos no llega a los dos segundos y medio. Esta marca temporal no se debe al algoritmo de generación de PDBs en sí, sino a la construcción de los grafos y tablas necesarias relacionadas con esos patrones. El tiempo se asemeja al de los algoritmos *combo*, es decir, los dos *Canonical* tardan casi lo mismo, y los dos *Zero-One* también. *iPDB*, sin embargo, requiere de mucho más tiempo en generar las PDBs. Por otra parte, los tiempos de poda y obtención de los conjuntos aditivos es despreciable.

Respecto al tiempo de búsqueda, *AlgHiking* mediante *Zero-One* tarda mucho menos tiempo que mediante *Canonical*, a pesar de que este segundo tenga una heurística más informada (dado que expande menos nodos). Esto puede deberse a que, al tener muchas PDBs (contando los conjuntos aditivos que se forman), se malgasta mucho tiempo en obtener una buena heurística. Sin embargo, parece ser que el conjunto de PDBs en este dominio funciona bastante bien mediante *Zero-One*. Una explicación de su buen funcionamiento es que, al haber tantas variables, es más difícil que una acción afecte a más de un patrón de manera simultánea, pudiendo aprovechar mejor el conjunto de patrones en ese orden de mayor a menor. Que la heurística sea mejor no significa que sea también la mejor alternativa. Por otra parte, los tiempos que aporta

Zero-One no llegan a ser tan bajos como los de *iPDB*, aunque no es tanta la diferencia. Es por eso por lo que el algoritmo que menos tiempo total consume es *AlgSpider* en *Zero-One*, seguido de *AlgSpider* en *Canonical*, después por el *combo* de *Canonical* y en último lugar *iPDB* y el otro algoritmo *combo* (siendo mejor este primero en problemas más complejos).

Observando la cantidad de memoria utilizada, *AlgSpider* en *Canonical* gasta menos que en *Zero-One*, aunque no es tan exagerada la diferencia. Por otra parte, *iPDB* solo emplea menos gasto de almacenamiento en los problemas 4 y 10, los cuales coinciden con dos de los cuatro problemas en los que *iPDB* utilizó variables distintas de las que se han utilizado al generalizar, como las *collect-card*. Por otra parte, *AlgSpider* en *Zero-One* utilizó más memoria que *iPDB* en tan solo 3 problemas más (contando los otros dos en los que supera a *Canonical*), y no por mucha diferencia.

Observando el número de nodos expandidos se puede llegar a apreciar la calidad de la heurística. *iPDB* supera en calidad de heurística a *AlgSpider* en *Zero-One* en todos los problemas, a excepción de dos problemas, y no con mucha diferencia. Sin embargo, en el caso de *Canonical*, *iPDB* solo le supera en los problemas 4, 10, 14 y 15. En los problemas 4, 10 y 14 la diferencia es mucho más significativa, y coincide con 3 de los 4 problemas donde se habían creado PDBs con variables distintas a las generalizadas, como las *collect-card*.

Y, por último, el algoritmo implementado mediante *Zero-One* resuelve el mismo número de problemas que el algoritmo *AlgSpider* con *Canonical*. Se puede vislumbar como el problema 11 tarda más de 2290 segundos, a pesar de haber establecido el límite de tiempo en 1800. Resulta que la comprobación del límite de tiempo se hace por cada iteración en la generación de las PDBs y en cada iteración en el proceso de búsqueda (por lo menos). En la iteración previa a encontrar la solución, el tiempo de ejecución no alcanzó los 1800 segundos, por lo que se procedió a ejecutar otra iteración. Cuando esta terminó, aunque se sobrapasase el tiempo máximo, se encontró lo solución, no abortando la ejecución. Siendo metodológico, podría descartarse esta solución. Por otra parte, *iPDB* es capaz de resolver dos problemas más que *AlgSpider*, el 13 y el 18. Es importante destacar que el problema 13 pertenece al cuarto problema donde se habían creado PDBs con variables distintas a las generalizadas, como las *collect-card*. Además, el coste de los planes es, obviamente, el mismo.

4.5.6 Extracción de conclusiones

Tras observar las anteriores comparaciones se pueden extraer conclusiones respecto al dominio *Spider*. En este apartado tan solo se realizarán aquellas más generales.

En caso de elegir entre el algoritmo *AlgSpider* mediante *Canonical* o mediante *Zero-One*, dado que resuelven el mismo número de problemas (sin descartar el problema en el que *AlgSpider* en *Canonical* sobrepasó los 1800 segundos), la mejor alternativa sería el de *Zero-One*. A pesar de que su heurística sea peor y emplee más memoria, la diferencia no es tan significativa. Merece la pena sacrificar un poco la calidad de la heurística con el fin de conseguir tiempos bastante más pequeños.

Respecto a la comparativa entre *AlgSpider* en *Zero-One* e *iPDB*, es más difícil establecer un mejor algoritmo. El punto positivo de AlgSpider es que el tiempo total es aún menor respecto a *iPDB*, al igual que la memoria empleada. Pero, por otra parte, *iPDB* elabora una mejor heurística en la mayor parte de los problemas (aunque no es muy significativa la diferencia), y es un algoritmo que se sabe adaptar bien a las condiciones del problema, llegando a resolver dos problemas más.

Generalizar en este dominio es bastante más difícil. Para resolver más problemas que *iPDB* es necesario hacer distinción entre estos para ver si aplicar un algoritmo u otro, lo cual no sería generalizar al completo, y complicaría el algoritmo a costa de unos mejores resultados.

Si tuviese que elegir uno de los dos algoritmos, me decantaría por *AlgSpider* en *Zero-One*, por el simple hecho de que resuelve los problemas en muchísimo menos tiempo, y el número de problemas adicionales resueltos por *iPDB* son tan solo 2 (siendo estos casos difíciles de generalizar por AlgSpider). Una buena práctica sería ejecutar un problema en *AlgSpider* mediante *Zero-One*, y si no se resuelve, optar por *iPDB*, que se adapta mejor a las condiciones del problema. Llegados hasta este punto podría decirse que *AlgSpider* se equipara con la potencia de *iPDB*, por lo que cumple con los requisitos para ser aceptado.

4.6 Dominio *Tetris*

4.6.1 Ejecución de los algoritmos

El primer paso es ejecutar los algoritmos seleccionados en el punto 4.1.2 Selección de algoritmos. Los resultados son los siguientes (*tablas 153 – 155 y 209 – 212*):

Tiempo total (s)	*iPDB (max time = 60)*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	116.675	7.97178	7.79474
P02	128.187	10.2394	10.4755
P03	-	-	-
P04	77.5581	3.45594	3.49084
P05	63.6368	9.74705	9.80438
P06	-	37.5652	37.3082
P07	-	-	-
P08	63.9123	3.22587	3.34126
P09	99.6254	7.69443	7.62612
P10	125.007	19.81	19.9162
P11	-	-	-
P12	65.8284	7.66022	7.74926
P13	-	324.978	324.846
P14	-	-	-
P15	88.3681	7.45877	7.69544
P16	-	-	-
P17	-	-	-

Tabla 153. Tiempo total – Tetris – Observación

Número nodos expandidos	*iPDB (max time = 60)*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	169372	6652	6652
P02	127104	9155	9155
P03	-	-	-
P04	17	7	7
P05	55472	3117	3117
P06	-	930042	930042
P07	-	-	-
P08	10	10	10
P09	29690	483	483
P10	1829129	141373	141373
P11	-	-	-
P12	285624	2410	2410
P13	-	8344225	8344225
P14	-	-	-
P15	1698	181	181
P16	-	-	-
P17	-	-	-

Tabla 154. Número nodos expandidos – Tetris – Observación

Coste de la solución	*iPDB (max time = 60)*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*
P01	30	30	30
P02	36	36	36
P03	-	-	-
P04	10	10	10
P05	30	30	30
P06	-	39	39
P07	-	-	-
P08	11	11	11
P09	21	21	21
P10	48	48	48
P11	-	-	-
P12	27	27	27
P13	-	56	56
P14	-	-	-
P15	19	19	19
P16	-	-	-
P17	-	-	-

Tabla 155. Coste de la solución – Tetris – Observación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.6.2 Observación de resultados

En primer lugar, se justifican los motivos por los cuales no se han introducido ciertos parámetros:

- ***pdb_max_size***: puesto que se ha introducido un parámetro relacionado con el tiempo máximo para obtener los patrones, no es del todo relevante modificar el tamaño máximo de las PDBs. Conviene dejar ejecutando *iPDB* hasta haber llegado al límite de tiempo de obtención de patrones. Merece la pena confiar en el valor por defecto de *pdb_max_size*. De todas formas, aumentar este valor tampoco iba a resultar muy beneficioso, pues las PDBs que se obtienen albergan muchas variables, debido a que estas pueden adoptar pocos valores, construyendo grafos no tan extensos. Alcanzar ese tamaño máximo de PDB es complicado.

- ***collection_max_size***: puesto que se ha introducido un parámetro relacionado con el tiempo máximo para obtener los patrones, no es del todo relevante modificar el tamaño máximo de la colección. Conviene dejar ejecutando *iPDB* hasta haber llegado al límite de tiempo de obtención de patrones. Merece la pena confiar en el valor por defecto de *collection_max_size*. De todas formas, aumentar este valor tampoco iba a resultar muy beneficioso, pues las PDBs que se obtienen albergan muchas variables, debido a que estas pueden adoptar pocos valores, construyendo grafos no tan extensos. Alcanzar ese tamaño máximo de colección es complicado.

- ***min_improvement***: dado que *iPDB* emplea una gran cantidad de tiempo en generar el conjunto de variables, bajar aún más el *improvement* no iba a ser beneficioso, al contrario. Por otra parte, subir dicho valor podría ser una buena idea, pero conviene más optar por un tiempo de ejecución fijo, y forzar a *iPDB* a ajustarse a ese tiempo. El conjunto de patrones extraídos de esta ejecución no es del todo malo.

- **La no inclusión de *max_time***: incorporar un valor por parámetro relacionado con el tiempo máximo para la obtención de PDBs era una necesidad, ya que tan solo en el primer problema *iPDB* no fue capaz de establecer un conjunto de PDBs tras una hora de ejecución. Un valor aceptable para este parámetro es de un minuto. De esta forma, se puede realizar una buena observación y crear un buen algoritmo.

- ***max_time_dominance_pruning* (en *iPDB*)**: el tiempo de poda es muy pequeño.

- ***max_time_dominance_pruning* (en *Canonical PDB*)**: el tiempo de poda es muy pequeño.

En *iPDB*, el tiempo de generación de las PDBs es muy grande respecto a los otros dos algoritmos, pero al acotarse a un minuto, tampoco es mucha la diferencia. Sin embargo, es cierto que, debido a la implementación de *iPDB*, algunas ejecuciones han alcanzado hasta los dos minutos. Esto se debe a que el algoritmo detiene la generación de PDBs hasta que en una iteración se consigue un tiempo mayor al establecido (exceptuando en la primera iteración, la cual puede detenerse antes de terminarla). Hasta que no termine esa iteración, no se detiene el algoritmo, pudiendo llegar hasta los dos minutos si hace falta. Por otra parte, no existen prácticamente diferencias entre los tiempos de generación de PDBs de *Canonical* y *Zero-One* sobre *combo*.

De nuevo, el tiempo de poda y de generación de conjuntos aditivos es despreciable. Respecto al tiempo de búsqueda, ambos algoritmos *combo* superan a *iPDB* con el límite de tiempo de 60 segundos. Además, las heurísticas de estos dos son mejores que la heurística de *iPDB*, pues expanden menos nodos. Las razones de esto es que en *combo* se ha recopilado un conjunto muy grande de variables en una sola PDB (o dos, siendo esta segunda PDB un patrón compuesto por una sola variable), e *iPDB* requiere de mucho tiempo de ejecución para llegar a formar esas PDBs tan grandes, incluso mejores que las de *combo*.

Respecto a la memoria utilizada, se puede contemplar cómo también los algoritmos *combo* dominan a *iPDB*. Además, estos resuelven dos problemas que *iPDB* no es capaz de resolver. Y, como resulta evidente, los costes de los planes generados son exactamente los mismos. Esto no implica que los planes sean iguales, pues puede darse el caso de que exista más de un plan meta con el mismo coste.

Dicho esto, debido a que el número de PDBs formado por *combo* es tan solo 1, merece más la pena usar *Zero-One* mediante este algoritmo *combo*. Tras este análisis, es fácil establecer cuál es el mejor algoritmo. Sin embargo, puede resultar interesante observar tanto los resultados de iPDB como los del algoritmo *combo*. Combinar las PDBs largas de *combo* y las variables consideradas como relevantes en *iPDB* podría llegar a potenciar la heurística, pudiendo superar el algoritmo *combo*.

Las características observadas de cada problema donde el algoritmo *iPDB* ha actuado, realizando previamente la poda y la creación de conjuntos aditivos, son las siguientes:

- **Problema 1**: una PDB con todas las metas y una no meta, todas variables *clear*. RESUELTO.

- **Problema 2**: *at_square* del cuadrado número 0 en todas sus posiciones con algunas variables meta. Genera más de un conjunto de PDBs. RESUELTO.

- **Problema 3**: genera más de un conjunto de PDBs. Alguna meta con un *clear* no meta.

- **Problema 4**: algunas metas con las posiciones el cuadrado número 0, con una posicion cuadrado no meta. RESUELTO.

- **Problema 5**: Igual que el problema 3, pero sin aditivas. RESUELTO.

- **Problema 6**: aparecen PDBs aditivas. Algunas metas en una PDB.

- **Problema 7**: aparecen PDBs aditivas. Unas cuantas PDBs, todas son metas. Sin embargo, una de ellas (PDB) tiene una *clear* no meta.

- **Problema 8**: dos PDBs, mezcla de metas y no metas. RESUELTO.

- **Problema 9**: una PDB con muchas metas y alguna no meta. RESUELTO.

- **Problema 10**: aparecen aditivas. Algunas metas en una PDB. RESUELTO.

- **Problema 11**: solo formas individuales, pero son aditivas.

- **Problema 12**: Igual que en el problema 1. RESUELTO.

- **Problema 13**: Aparecen PDBs aditivas. Algunas metas en una PDB.

- **Problema 14**: igual que en el problema 13, pero la PDB es más pequeña (el número de variables es muy grande y no da tiempo a avanzar *iPDB*).

- **Problema 15**: Igual que en el problema 1, pero son dos *clear* no metas. También en todas tras la poda hay 3 PDBs, así que habrá seguramente PDBs aditivas. RESUELTO.

- **Problema 16**: Aparecen PDBs aditivas. Pocas varibles metas junto a no metas (muy pocas variables por PDB. El fichero *output.sas* se compone de muchas variables).

- **Problema 17**: Igual que en el problema 11.

Las características observadas de cada problema donde el algoritmo *combo* ha actuado son las siguientes:

- **Problema 1**: 19 variables desde el final. RESUELTO.

- **Problema 2**: Igual que problema 1. RESUELTO.

- **Problema 3**: Igual que en el problema 1, pero con una PDB individual con tan solo una variable meta.

- **Problema 4**: Igual que en el problema 1. RESUELTO.

- **Problema 5**: Igual que en el problema 1. RESUELTO.

- **Problema 6**: Igual que en el problema 1. RESUELTO.

- **Problema 7**: Igual que en el problema 3.

- **Problema 8**: igual que en el problema 1. RESUELTO.

- **Problema 9**: igual que en el problema 1. RESUELTO.

- **Problema 10**: igual que en el problema 1. RESUELTO.

- **Problema 11**: igual que en el problema 3.

- **Problema 12**: igual que en el problema 1. RESUELTO.

- **Problema 13**: igual que en el problema 1. RESUELTO.

- **Problema 14**: igual que en el problema 3.

- **Problema 15**: igual que en el problema 1. RESUELTO.

- **Problema 16**: igual que en el problema 1.

- **Problema 17**: igual que en el problema 3.

Se puede ver cómo predominan las variables *clear*, tanto las relacionadas con predicados meta como las que no, estando estas variables meta en orden consecutivo. *iPDB* no aporta mucha más información. Sin embargo, *combo* señala que es relevante crear PDBs extensas, con unas 19 variables a lo sumo. Esto no es suficiente como para crear un algoritmo que pretenda superar las marcas temporales del algoritmo *combo*.

Observando el dominio en sí, se puede ver cómo las fichas del *Tetris* se mueven tanto horizontal como verticalmente. Dicho esto, se ha pensado en implementar, con el fin de generalizar, un algoritmo que cree PDBs con las siguientes características:

- Una PDB que actúe como el algoritmo *combo*, es decir, que reúna 19 variables *clear*. Sin embargo, dado que las fichas se pueden mover en horizontal, se cogerán variables de forma ordenada empezando por la esquina superior izquierda, con el propósito de terminar en la esquina inferior derecha de la franja relacionada con los *clear* de meta (dejar liberada la parte superior del tablero, es decir, mover las piezas abajo). En caso de que se incluyan todas en una sola PDB, se completará con *clear* no relacionadas con meta hasta alcanzar las 19 variables, siguiendo el mismo orden. Sin embargo, en caso de que al completar las 19 variables aún siga habiendo variables *clear* meta, se introducirá otra PDB de 19 variables, pero empezando por la antepenúltima variable de esta PDB completada, aunque se termine llenando de variables *clear* no meta (o llegando al punto de que ya no haya más variables *clear* que introducir). Esto se hace así

para que entre PDBs haya variables en común. De esta forma, nos aseguramos de que todos los patrones meta se encuentran en una PDB al menos.

- Sabiendo que las fichas en el *Tetris* caen, podría resultar interesante crear una PDB por columna, introduciendo en esta todos los *clear* de las posiciones de esa misma columna. De esta forma, se añaden todos los *clear* en al menos una PDB, y toda PDB tiene una variable meta al menos.

No se tendrán en cuenta PDBs individuales (con una sola variable) porque todas las casillas están unidas entre sí, y aporta más información, en principio, todas estas variables unidas que por separado. Estas PDBs individuales no aportarían nada a la heurística. Además, al igual que se ha hecho en el resto de los dominios, primero se colocan los patrones más grandes, y luego los patrones individuales, con el fin de favorecer a *Zero-One PDB*, aunque para *Canonical PDB* no sea relevante la ordenación.

4.6.3 Implementación y ejecución del algoritmo

En este punto se implementará el algoritmo de acorde a la generalización realizada en el anterior punto. El *script* implementado se comenta en el punto 3.3.7 *Script* de algoritmo para *Tetris*.

Los resultados, dado que se van a comparar posteriormente con el resto de los algoritmos, se muestran en conjunto (*tablas 156 – 158 y 209 – 212*):

Tiempo total (s)	*iPDB (max time* = 60)	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgTetris (Canonical PDB)*	*AlgTetris (Zero-One PDB)*
P01	116.675	7.97178	7.79474	8.16044188	8.36718188
P02	128.187	10.2394	10.4755	10.26613210	10.41943210
P03	-	-	-	-	-
P04	77.5581	3.45594	3.49084	0.44831291	0.45130991
P05	63.6368	9.74705	9.80438	9.86774204	10.12237204
P06	-	37.5652	37.3082	24.22178288	24.34918288

P07	-	-	-	-	-
P08	63.9123	3.22587	3.34126	0.41998802	0.41893902
P09	99.6254	7.69443	7.62612	8.36833290	8.36374290
P10	125.007	19.81	19.9162	19.03626089	18.95806089
P11	-	-	-	-	-
P12	65.8284	7.66022	7.74926	8.09096503	8.10703503
P13	-	324.978	324.846	212.04270498	226.56870498
P14	-	-	-	-	-
P15	88.3681	7.45877	7.69544	8.07321586	7.85987586
P16	-	-	-	-	-
P17	-	-	-	288.63949498	280.74249498

Tabla 156. Tiempo total – Tetris – Evaluación

Número nodos expandidos	***iPDB (max time = 60)***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTetris (Canonical PDB)***	***AlgTetris (Zero-One PDB)***
P01	169372	6652	6652	1501	1501
P02	127104	9155	9155	4431	4431
P03	-	-	-	-	-
P04	17	7	7	7	7
P05	55472	3117	3117	1218	1218
P06	-	930042	930042	437305	437305

P07	-	-	-	-	-
P08	10	10	10	10	10
P09	29690	483	483	17	17
P10	1829129	141373	141373	90540	90540
P11	-	-	-	-	-
P12	285624	2410	2410	1642	1642
P13	-	8344225	8344225	5143380	5143380
P14	-	-	-	-	-
P15	1698	181	181	102	102
P16	-	-	-	-	-
P17	-	-	-	5212715	5212725

Tabla 157. Número nodos expandidos – Tetris – Evaluación

Coste de la solución	*iPDB (max time = 60)*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgTetris (Canonical PDB)*	*AlgTetris (Zero-One PDB)*
P01	30	30	30	30	30
P02	36	36	36	36	36
P03	-	-	-	-	-
P04	10	10	10	10	10
P05	30	30	30	30	30
P06	-	39	39	39	39

P07	-	-	-	-	-
P08	11	11	11	11	11
P09	21	21	21	21	21
P10	48	48	48	48	48
P11	-	-	-	-	-
P12	27	27	27	27	27
P13	-	56	56	56	56
P14	-	-	-	-	-
P15	19	19	19	19	19
P16	-	-	-	-	-
P17	-	-	-	66	66

Tabla 158. Coste de la solución – Tetris – Evaluación

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.6.4 Pruebas del algoritmo

Estas pruebas consisten en seleccionar 3 problemas. Es favorable que ofrezcan resultados diferentes entre sí, para probar mejor el algoritmo. Se compara la salida esperada con la obtenida, y si coinciden, se podría verificar su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validaría.

Se van a observar los problemas 1, 13 y 17. Un problema con 12 predicados meta (<19 y tablero pequeño), otro con 16 (<19 y tablero grande) y otro con 20 (>19).

- El algoritmo implementado sobre el problema 1 muestra las siguientes PDBs:

[411, 413, 414, 415, 416, 417, 418, 419, 420, 421, 422, 423, 424, 425, 426, 427, 428, 429, 430]

[408, 411, 413, 419, 423, 425]

[409, 415, 417, 421, 427, 429]

[410, 416, 418, 422, 428, 430]

[407, 412, 414, 420, 424, 426]

407 – 430: variables *clear*.

Cada variable *clear* está relacionada con una casilla del tablero de juego. A continuación, se muestra a qué posiciones corresponden las anteriores variables. Las casillas con un tono más oscuro hacen referencia a variables meta (*tabla 159*):

419	421	422	420
423	427	428	424
425	429	430	426
413	417	418	414
411	415	416	412
408	409	410	407

Tabla 159. Tablero de juego de la primera prueba

- El resto de las pruebas se encuentran en la Sección 9 Anexo.

En el primer problema se puede ver una PDB con 19 variables, correspondientes a las 12 variables meta, y a las variables adicionales 413, 417, 418, 414, 411, 415 y 416. En el problema 13, la PDB *larga* se corresponde con las 16 variables meta, y con las variables adicionales 1173, 1177 y 1178. Por último, en el problema 17 aparecen dos PDBs *largas*, una de ellas empezando en la casilla con la variable 1830 y recogiendo variables por filas hasta acabar en la casilla con el 1849, y la otra desde la variable meta que faltaba (1840) y terminando en la casilla con el 1812.

También aparece otra PDB adicional por cada columna del problema (que siempre son 4) con todas las variables de esa columna. Es por eso por lo que esas PDBs tienen tamaño 6 en el problema 1, 8 en el problema 13 y 10 en el problema 17. Además, todas estas PDBs creadas se ordenan de menor a mayor.

Como se puede comprobar, el algoritmo funciona correctamente. Queda, por tanto, verificado y validado, pues lo importante del algoritmo, dado que es sencillo, son las PDBs que construye. Dadas las características del *Script*, un análisis más exhaustivo relacionado con una metodología más dura no tendría sentido.

4.6.5 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con el resto de los algoritmos.

En primer lugar, respecto al tiempo de generación de las PDBs, el algoritmo implementado no requiere de mucho tiempo. En el peor de los casos no alcanza el medio minuto, siendo menor que el que muestra *iPDB* con el tiempo máximo de un minuto. Tanto en *Zero-One* como en *Canonical* los resultados son parecidos. Esto no se debe al algoritmo de generación de PDBs en sí, sino a la construcción de los grafos y tablas necesarias relacionadas con esos patrones. Comparando los tiempos de *AlgTetris* con *combo* se puede ver cómo este segundo proporciona, por lo general, tiempos menores, lo cual tiene sentido, pues las PDBs que se forman son menores. En el peor de los casos el algoritmo implementado tarda en construir las PDBs el doble de tiempo que *combo*. Por otra parte, los tiempos de poda y obtención de los conjuntos aditivos es despreciable.

Analizando el tiempo de búsqueda, se puede ver cómo *AlgTetris* tarda menos que los *combo*. Entre *Canonical* y *Zero-One*, ambos algoritmos tardan prácticamente lo mismo, con algunas ligeras diferencias. Al observar el número de nodos expandidos se puede apreciar cómo es inferior en AlgTetris, pero no existe una diferencia significativa entre *Canonical* y *Zero-One*. Es más, las cantidades son las mismas a excepción del último problema, en el que la diferencia es tan solo de 10 nodos más. La heurística de *AlgTetris* en *Canonical* es mejor que la de *Zero-One*, pero por muy poco. Al parecer, *Zero-One* funciona bien sobre estas PDBs, pues es posible que muy pocas acciones actúen de forma simultánea sobre dos PDBs, aprovechándolas más.

Respecto a la memoria *AlgTetris* vuelve a tomar la delantera frente a *combo*, pues su heurística es mejor. Pero entre los dos *AlgTetris* no existen diferencias notables, pues *Canonical* es ligeramente mejor que *Zero-One* en este aspecto. Por último, el coste de la solución de todos los algoritmos es el mismo, como cabría esperar. Se puede observar cómo *AlgTetris* resuelve un problema más que los algoritmos *combo*.

4.6.6 Extracción de conclusiones

En este caso, elegir un algoritmo no es una tarea complicada. *AlgTetris* es superior tanto a *iPDB* como a los algoritmos *combo*. Sin embargo, en caso de elegir entre *AlgTetris* en *Canonical* y entre *Zero-One*, debido a la calidad de la heurística sería una mejor opción *Canonical*, aunque la diferencia de mejora sea tan pequeña.

4.7 Independiente del dominio

4.7.1 Observación de características similares entre las PDBs relevantes de cada dominio

A partir de las ejecuciones y observaciones realizadas durante el desarrollo de las heurísticas dependientes del dominio, se extraerán sus características similares respecto a las variables utilizadas. Para ello, dado que el futuro algoritmo a implementar no comprenderá la semántica, se identificarán estas similitudes mediante la ubicación de cada una de estas variables *relevantes* en la codificación de su dominio correspondiente. Dicho esto, se identificarán aquellos predicados incluidos en estas variables consideradas como *relevantes*, así como las metas que aparecen en los problemas.

En primer lugar, se recordará cuáles son las metas y variables importantes de cada dominio:

– *Snake*:

- ⇨ Metas: *ispoint*.
- ⇨ Variables vistas como importantes: *spawn*, *headsnake*, *ispoint*.

– *Termes*:

- ⇨ Metas: la altura (*height*) de todas las casillas del mapa, excepto del *depot*.
- ⇨ Variables vistas como importantes: *height*, *at* y *has-block*.

– *Hiking*:

- ⇨ Metas: variables *walked*.
- ⇨ Variables vistas como importantes: variables *walked*, *at_person*, *at_tent* y *up* y *down*.

– *Spider*:

- ⇨ Metas: variables *on* y los *clear* de *deal* y los de *pile*.

- ⇨ Variables vistas como importantes: las metas *on* y *clear* (*pile* y *deal*), *currently-collecting-deck*, *current-deal* y *currently-dealing*.

-***Tetris***:

- ⇨ Metas: *clear*.
- ⇨ Variables vistas como importantes: *clear*.

Llegados a este punto, parece ser que lo más sensato sería fijarse en las acciones donde aparece algún predicado meta. Dado que el objetivo del problema es conseguir esos predicados meta, el resto de los predicados de esa misma acción donde aparecen podrían ser útiles. Dicho esto, el siguiente objetivo es encontrar una posible correlación entre las variables de las PDBs de los algoritmos anteriormente implementados y los predicados de las acciones donde hay un predicado meta.

Dicho esto, los predicados extraídos de cada acción pueden ser de 4 tipos:

- **Conjunto A**: aquellos predicados que aparecen en las precondiciones de las acciones que originan un cambio de un predicado meta en los efectos.

- **Conjunto B**: aquellos predicados que aparecen en los efectos de las acciones junto a un predicado meta.

- **Conjunto C**: aquellos predicados que aparecen en los efectos de las acciones que tienen en las precondiciones un predicado meta.

- **Conjunto D**: aquellos predicados que aparecen en las precondiciones de las acciones junto a un predicado meta.

Se pretende realizar un estudio extenso por fuerza bruta. Pero, antes que nada, estaría bien entender la amplitud del problema a resolver, y cuestionarse si realizar un estudio sin analizar previamente los conjuntos es una buena idea. Dicho esto, se extraerán los predicados de cada conjunto (*A, B, C* y *D*) para observar si es viable afrontar el estudio de esta forma. Antes de nada, es necesario tener en cuenta unas cuestiones:

- La primera es que se ignorarán las sentencias *when*, es decir, se escogerán los predicados cumpliendo o no la condición *when*. Lo más sensato sería duplicar esas acciones por cada sentencia *when*, una que tenga las precondiciones del *when*, pudiendo realizar los efectos adicionales incluidos en esa sentencia, y otra acción que no las cumpla y no aplique dichos efectos. Sin embargo, en el presente estudio los predicados que aparecen en esas sentencias no son relevantes. Futuros estudios podrían considerarlas particionando las acciones, tal y como se acaba de decir.

- La segunda cuestión se relaciona con la interpretación de las metas. Si en un problema hubiese 2 predicados distintos en las metas, se crearían 3 conjuntos *A, B, C* y *D*, uno por cada meta Y otro considerando todas las metas a la vez. Esto puede hacerse dado que todos los problemas tienen los mismos tipos de predicados en un mismo dominio.

- No se tendrá en cuenta el signo de los predicados meta en las acciones. Es decir, si un problema tuviese un predicado indicando que no debe aparecer cierta instanciación de un predicado, *(not (<nombre del predicado> <lista de objetos>))*, bastará con completar los conjuntos *A, B, C* y *D* sin fijarse en si el predicado meta aparece negado (con *not*) o no.

- Los predicados de las acciones de un mismo conjunto no se agruparán, sino que se mantendrán separados. Podría decirse que *A, B, C* y *D* son conjuntos de conjuntos de predicados.

Los conjuntos en cada dominio son los siguientes:

– *Snake*:

- ⇨ A:
  - *headsnake, blocked, ispoint, spawn*.
  - *headsnake, blocked, ispoint, spawn*.
- ⇨ B:
  - *blocked, headsnake, nextsnake, ispoint, spawn*.
  - *blocked, headsnake, nextsnake, ispoint*.
- ⇨ C:
  - *headsnake, blocked, nextsnake, tailsnake*.
  - *blocked, headsnake, nextsnake, ispoint, spawn*.
  - *blocked, headsnake, nextsnake, ispoint*.
- ⇨ D:
  - *headsnake, tailsnake, nextsnake, blocked, ispoint*.
  - *headsnake, blocked, ispoint, spawn*.
  - *headsnake, blocked, ispoint, spawn*.

– *Termes*:

- ⇨ A (con *height*):
  - *height, at, has-block.*
  - *height, at, has-block.*
- ⇨ B (con *height*):
  - height, has-block.
  - height, has-block.
- ⇨ C (con *height*):
  - *at.*

- *at.*
- *at.*
- *height, has-block.*
- *height, has-block.*

⇨ D (con *height*):
- *at, height.*
- *at, height.*
- *at, height.*
- *at, height, has-block.*
- *at, height, has-block,*

⇨ A (con *has-block*):
- *height, at, has-block.*
- *height, at, has-block.*
- *has-block, at.*
- *has-block, at.*

⇨ B (con *has-block*):
- *height, has-block.*
- *height, has-block.*
- *has-block.*
- *has-block*

⇨ C (con *has-block*):
- *height, has-block.*
- *height, has-block.*
- *has-block.*
- *has-block.*

⇨ D (con *has-block*):
- *height, at, has-block.*
- *height, at, has-block.*
- *at, has-block.*
- *at, has-block.*

⇨ A (con *height* y *has-bloc*k):
- *height, at, has-block.*
- *height, at, has-block.*

⇨ B (con *height* y *has-block*):
- *height, has-block.*
- *height, has-block.*

⇨ C (con *height* y *has-block*):
- *height, has-block.*
- *height, has-block.*

⇨ D (con *height* y *has-block*):
- *height, at, has-block.*
- *height, at, has-block.*

– *Hiking*:

- A:
  - *at_tent, up, at_person, walked.*
- B:
  - *at_person, walked.*
- C:
  - *at_person, walked.*
- D:
  - *at_tent, up, at_person, walked.*

– *Spider*:

- A (con *on*):
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu-*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*
- B (con *on*):
  - *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
  - *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, make-movable, currently-updating-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, in-play, part-of-tableu, movable, collect-card.*
  - *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*
- C (con *on*):
  - *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*

- *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, make-movable, currently-updating-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *part-of-tableau, make-part-of-tableau.*
- *movable, make-unmovable.*
- *movable, make-unmovable, currently-updating-unmovable.*
- *movable, make-unmovable, currently-updating-unmovable, currently-updating-part-of-tableau.*
- *make-movable, movable.*
- *make-movable, movable, currently-updating-movable.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

⇨ D (con *on*):

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
- *currently-updating-part-of-tableau, currently-collecting-deck, on, make-part-of-tableau, part-of-tableau.*
- *currently-updating-unmovable, currently-collecting-deck, currently-updating-part-of-tableau, on, make-movable, movable.*
- *currently-updating-unmovable, currently-collecting-deck, currently-updating-part-of-tableau, on, make-unmovable.*
- *currently-updating-unmovable, currently-collecting-deck, currently-updating-part-of-tableau, on, make-unmovable, movable.*
- *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, make-movable.*
- *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, make-movable.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*

- ⇨ A (con *clear*):
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*
- ⇨ B (con *clear*):
  - *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, in-play, part-of-tableu, movable, collect-card.*
  - *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*
- ⇨ C (con *clear*):
  - *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
  - *currently-dealing, current-deal.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *part-of-tableau, make-part-of-tableau, currently-updating-part-of-tableau.*
  - *currently-collecting-deck, collect-card.*
- ⇨ D (con *clear*):
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableu.*

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
- *currently-updating-part-of-tableau, currently-collecting-deck, make-part-of-tableau, part-of-tableau, clear.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, clar, in-play.*

⇨ A (con *clear* y *on*):

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*

⇨ B (con *clear* y *on*):

- *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

- ⇨ C (con *clear* y *on*):
  - *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
  - *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- ⇨ D (con *clear* y *on*):
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableu.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
  - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*

– *Tetris*:

- ⇨ A:
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
- ⇨ B:
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
- ⇨ C:
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*

- ⇨ D:
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*

Observando vagamente los conjuntos *A, B, C* y *D* en su proceso de creación se han alcanzado las siguientes dos conclusiones que guiarán la investigación:

- La cantidad de conjuntos es muy extensa. No es necesario investigar todos ellos, ya que un análisis previo podría servir de ayuda para filtrar aquellos que no vaticinan buenos resultados.

- En los procesos de investigación no resulta siempre la mejor idea analizar absolutamente todas las posibilidades, pues un conocimiento extenso de la situación y buenos razonamientos podrían llegar a acortar la metodología, incrementando la calidad de la solución.

Teniendo todo esto en cuenta, se ha decidido tomar las siguientes medidas que guiarán la investigación hasta el final:

- La investigación de este punto se basará en un análisis empírico, pero decidiendo concienzudamente qué analizar, justificando siempre la respuesta.

- Solo se utilizarán los conjuntos *A* y *B*, pues no tiene demasiado sentido observar los predicados que aparecen en las acciones donde aparece un predicado meta en las precondiciones. Si lo que se quiere conseguir es alcanzar dichas metas, lo más lógico es que aparezcan en los efectos. Esta decisión reduce la carga de trabajo de esta metodología a la mitad, aproximadamente. Resulta más intuitivo pensar que, para alcanzar esa meta, haya que considerar ciertos predicados que estén en las precondiciones de las acciones que originen ese cambio. Además, la observación de predicados situados junto a los predicados meta en los efectos de estas acciones también podría resultar una buena idea, ya que las metas suelen componerse de predicados iguales. En cuanto algunas condiciones metas se empezasen a alcanzar, se generarían simultáneamente estas instanciaciones de los predicados, pues ambos están en los efectos de las acciones que se han ejecutado. Considerar también estos predicados podría orientar la heurística de una mejor manera. Todo depende de los resultados de la investigación.

- Dado que se ha reducido el número de conjuntos un 50%, sería apropiado intensificar la investigación de los dos conjuntos elegidos. Dicho esto, se ha decidido tener en cuenta el signo de los predicados meta en las acciones. Es decir, si en un problema hubiese un predicado indicando que no debe aparecer cierta instanciación de un predicado, *(not (<nombre del predicado> <lista de objetos>))*, será necesario completar los conjuntos *A* y *B* fijándose si el predicado meta aparece negado (con *not*) o no.

Sin embargo, no se tendrá en cuenta el signo de los predicados recopilados en los conjuntos *A* y *B*, tan solo a la hora de identificar los predicados meta. Esto se ha decidido hacerlo así porque, dentro de una misma variable, las instancias de los predicados suelen ir acompañadas de las mismas instancias de predicados negadas (pues son dos condiciones que no pueden suceder a la vez). Llegar a ser tan restrictivo en cuanto al signo en estas ocasiones no ocasionaría ninguna diferencia positiva, todo lo contrario, se descartarían muchos más predicados a costa de ninguna mejora en caso de realizar operaciones entre conjuntos posteriormente.

- Además, como previamente es difícil de determinar si al realizar esta investigación teniendo en cuenta el signo se van a alcanzar buenos resultados, se ha decidido también no tener en cuenta el signo.

- Se continuarán ignorando las sentencias *when* tal y como se dijo anteriormente, pues en los dominios elegidos los predicados que aparecen en esas sentencias no son relevantes, no siendo capaces de determinar finalmente si es mejor ignorarlas o considerarlas en cualquier otro dominio distinto. Sin embargo, se puede afirmar con certeza que, si en la investigación actual se obtienen buenos resultados, podría llegarse a la conclusión de que duplicar las acciones que las contienen por cada sentencia *when* que aparezca en esta (creando una que tenga las precondiciones del *when*, pudiendo realizar los efectos adicionales incluidos en esa sentencia, y otra acción que no las cumpla y no aplique dichos efectos) dará buenos resultados, ya que ambos dominios creados serían equivalentes entre sí. Esto sería cierto dado que las precondiciones y los efectos de los *when* no tienen ni predicados *relevantes* ni instanciaciones de predicados meta.

- Se analizarán las metas de otra forma distinta a la expuesta anteriormente expuesta, no teniendo en cuenta la combinatoria entre metas. Por ejemplo, si en un problema hubiese 2 predicados distintos en las metas, se crearían 2 conjuntos A y B *teniendo en cuenta el signo* y otros *A* y *B sin tener en cuenta el signo*, uno por cada meta. Esto puede hacerse dado que todos los problemas tienen los mismos tipos de predicados en un mismo dominio. Analizar las dos metas a la vez es equivalente a realizar operaciones de conjuntos entre los diferentes predicados obtenidos por cada meta, lo cual crea redundancia sobre la investigación. Además, no tienen por qué obtenerse instanciaciones de distintos predicados meta a la vez.

- De igual forma, los predicados de las acciones de un mismo conjunto no se agruparán, sino que se mantendrán separados. Podría decirse que *A* y *B* son conjuntos de conjuntos de predicados. Así se permitirá un análisis más exhaustivo.

- Aunque en *Spider* es redundante que haya metas con predicados *clear* y *on*, se tendrán en cuenta todas las instanciaciones de predicados que estén en la meta, pues un algoritmo simple no sería capaz de determinar esta redundancia. Se dice que es redundante ya que todas las cartas están descartadas (predicado *on discard*) si y solo si no hay cartas en la pila ni en el *deal* (*clear pile* y *clear deal*). De todos modos, esta información redundante ayuda a guiar mejor a los algoritmos, creando mejores heurísticas.

- Aunque se ha hecho distinción entre predicados teniendo en cuenta el signo (*not*) y predicados sin tener en cuenta el signo, no se ha considerado la posibilidad de que un mismo predicado pueda tener varios tipos de objetos. Entre todos los dominios utilizados solo sucede en *Spider*; por ejemplo, con el predicado *clear*, que puede venir acompañado de una carta, *pile* o *deal*. Dicho esto, si en las metas aparece dicho predicado *clear* acompañado de *pile* y de *deal*, tan solo considerarán aquellas acciones en donde los efectos esté este predicado actuando sobre este tipo de objetos en concreto. La principal razón por la cuál esto no se ha hecho así es para no restringir demasiado las condiciones. Al fin y al cabo, es interesante saber la localización de una carta, aunque no esté descartada (*on*), si una carta puede moverse porque no hay ninguna encima (*clear*), y si se ha vaciado la pila y el *deal* (*clear*). Y la segunda razón es que el hecho de tener en cuenta los tipos de objetos que acompañan a los predicados puede dar lugar a otro estudio comparativo, con el fin de mostrar si es relevante especificar hasta ese detalle o actuar de una forma más general. Especificar tanto puede conllevar a descartar tantos predicados y variables dando lugar a algoritmos y PDBs pobres y poco potentes.

- Tras obtener cada uno de los conjuntos se realizarán operaciones sobre conjuntos en el caso de no tener en cuenta el signo y teniendo en cuenta el signo, por cada uno de los dominios. Estableciendo finalmente qué combinación se aproxima mejor a las variables obtenidas en la investigación dependiente del dominio.

- En caso de tener que tomar una decisión en el transcurso de esta investigación, se analizará in situ.

- Una vez se haya llegado a obtener una relación entre las metas y las variables *relevantes*, se decidirá en ese momento cómo avanzar la investigación en función de los resultados obtenidos.

Los conjuntos en cada dominio son los siguientes:

– *Snake*:

- ⇨ A (*sin signo*):
  - *headsnake, blocked, ispoint, spawn.*
  - *headsnake, blocked, ispoint, spawn.*
- ⇨ B (*sin signo*):
  - *blocked, headsnake, nextsnake, ispoint, spawn.*
  - *blocked, headsnake, nextsnake, ispoint.*
- ⇨ A (*con signo*):
  - *headsnake, blocked, ispoint, spawn.*
  - *headsnake, blocked, ispoint, spawn.*
- ⇨ B (*con signo*):
  - *blocked, headsnake, nextsnake, ispoint, spawn.*
  - *blocked, headsnake, nextsnake, ispoint.*

– *Termes*:

- ⇨ A (*sin signo* y con *height*):
  - *height, at, has-block.*
  - *height, at, has-block.*
- ⇨ A (*sin signo* y con *has-block*):
  - *height, at, has-block.*
  - *height, at, has-block.*
  - *has-block, at.*
  - *has-block, at.*
- ⇨ B (*sin signo* y con *height*):
  - *height, has-block.*
  - *height, has-block.*
- ⇨ B (*sin signo* y con *has-block*):
  - *height, has-block.*
  - *height, has-block.*
  - *has-block.*
  - *has-block.*
- ⇨ A (*con signo* y con *height*):
  - *height, at, has-block.*
  - *height, at, has-block.*
- ⇨ A (*con signo* y con *has-block*):
  - *height, at, has-block.*
  - *has-block, at.*
- ⇨ B (*con signo* y con *height*):
  - *height, has-block.*
  - *height, has-block.*
- ⇨ B (*con signo* y con *has-block*):

- *height, has-block.*
- *has-block.*

– *Hiking*:

⇨ A (*sin signo*):
- *at_tent, up, at_person, walked.*

⇨ B (*sin signo*):
- *at_person, walked.*

⇨ A (*con signo*):
- *at_tent, up, at_person, walked.*

⇨ B (*con signo*):
- *at_person, walked.*

– *Spider*:

⇨ A (*sin signo* y con *on*):
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu-*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*

⇨ A (*sin signo* y con *clear*):
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*

⇨ B (*sin signo* y con *on*):

- *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
- *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, make-movable, currently-updating-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

⇨ B (*sin signo* y con *clear*):

- *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

⇨ A (*con signo* y con *on*):

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.*
- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu-*

- *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.*

⇨ A (*con signo* y con *clear*):

- currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau.
- currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, part-of-tableu, on.
- currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, movable, in-play, clear, on.
- currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, collect-card, on, in-play, part-of-tableu.
- currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, collect-card, in-play, part-of-tableu.

⇨ B (*con signo* y con *on*):

- *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
- *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, make-movable, currently-updating-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

⇨ B (*con signo* y con *clear*):

- *on, clear, in-play, part-of-tableu, movable, currently-updating-unmovable, make-unmovable.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, currently-updating-movable, make-movable, currently-updating-part-of-tableau, make-part-of-tableau.*
- *on, clear, in-play, part-of-tableu, movable, collect-card.*
- *on, clear, in-play, part-of-tableu, movable, collect-card, currently-collecting-deck, make-movable, currently-updating-movable.*

– *Tetris*:

- ⇨ A (*sin signo*):
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
- ⇨ B (*sin signo*):
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
- ⇨ A (*con signo*):
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
- ⇨ B (*con signo*):
  - *clear, at_square.*
  - *clear, at_two.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*
  - *clear, at_right_l.*

Una vez obtenido cada conjunto, se extraen las siguientes consideraciones y se tendrán en cuenta las siguientes premisas:

- Dadas las condiciones de los dominios, es equivalente tener en cuenta el signo que no tenerlo en *Snake*, *Hiking*, *Spider* y *Tetris*. Por esa razón, solo se analizará dicha alternativa en el dominio Termes, debido las condiciones de este dominio. Sin embargo, el algoritmo implementado actuará de la misma forma en cualquier dominio.

- Los operadores de conjuntos que se van a utilizar son la unión (U) y la intersección (∩), ya que son los más básicos. El operador *not* no tendría sentido, pues no parece lógico negar los predicados de un conjunto que se consideran de antemano relevantes,

obteniendo el resto de los predicados del dominio que se habían descartado previamente.

- Dado que A y B son conjuntos de conjuntos de predicados, también podrían realizarse las operaciones de unión e intersección de forma interna.

- Las posibles combinaciones entre conjuntos que se van a estudiar son:

- ⇨ UA U UB.
- ⇨ UA ∩ UB.
- ⇨ ∩A U UB.
- ⇨ ∩A ∩ UB.
- ⇨ UA U ∩B.
- ⇨ UA ∩ ∩B.
- ⇨ ∩A U ∩B.
- ⇨ ∩A ∩ ∩B.
- ⇨ UA.
- ⇨ UB.
- ⇨ ∩A.
- ⇨ ∩B.

- No se van a realizar otro tipo de operaciones ni agrupaciones entre los conjuntos, pues puede complicarse de gran manera debido a la explosión combinatoria que se puede generar. Como trabajo futuro podría implementarse un algoritmo inspirado en programación genética y en teoría de conjuntos, con el fin de obtener una operación de conjuntos que mejor se aproxime a las variables relevantes.

- En el caso de que existan varios predicados meta, se pueden tomar dos posibles alternativas: unirlas al principio o unirlas al final. Es difícil decantarse por una de ellas a priori, por lo que se investigarán los resultados de ambas. En caso de que, tras la investigación, se decida que es mejor unirlas al final o al principio, esto no implica que futuras investigaciones no sean acertadas cuando se cuestionen esa afirmación, ya que este proyecto puede considerarse como una base inicial para futuras investigaciones que sigan la misma línea. Como puede resultar confuso, se explicarán las diferencias que radican entre unirlas al principio y unirlas al final:

- ⇨ Unir al principio significa hacer una unión de todos los predicados de distintas metas, formando un único conjunto **antes** de aplicar las operaciones. En caso de hacer una intersección sobre ese mismo conjunto, implicaría que los predicados *relevantes* deberían aparecer **en el conjunto de todas** las acciones con predicados meta en las acciones.
- ⇨ Unir al final significa hacer una unión de todos los predicados de distintas metas, formando un único conjunto **después** de aplicar las operaciones. Podría decirse que requiere de un poco más de tiempo de cómputo, pues se hace una operación

de conjuntos por cada predicado meta distinto. En caso de hacer una intersección sobre ese mismo conjunto, implicaría que se tienen en cuenta los predicados *relevantes* que aparecen **en cada una** de las acciones con al menos una meta en los efectos.

Plantear hacer una intersección entre los conjuntos formados por cada meta no se baraja como una alternativa en el proyecto, pues forman mayoritariamente operaciones entre conjuntos equivalentes a las que se pueden crear con las ya opciones elegidas. Por ejemplo, hacer una intersección entre los conjuntos creados mediante cada meta y después otra intersección entre cada elemento del conjunto sería equivalente a realizar una unión entre las metas y después una intersección entre todos los elementos.

- Finalmente, se escogerá aquella operación con conjuntos que mejor se aproxime a los predicados *relevantes*. Aunque sería más apropiado terminar eligiendo una operación **simple** entre conjuntos, es decir, aquella que solo utiliza un conjunto. Se antepondrá una operación simple ante una compleja si proporcionan el mismo resultado, ya que en muchas situaciones favorecen y se premian las soluciones más sencillas.

Dicho esto, los resultados de las operaciones son las siguientes:

– *Snake*:

- ⇨ UA U UB - *headsnake, blocked, ispoint, spawn, nextsnake.*
- ⇨ UA ∩ UB - *headsnake, blocked, ispoint, spawn.*
- ⇨ ∩A U UB - *headsnake, blocked, ispoint, spawn, nextsnake.*
- ⇨ ∩A ∩ UB - *headsnake, blocked, ispoint, spawn.*
- ⇨ UA U ∩B - *headsnake, blocked, ispoint, spawn, nextsnake.*
- ⇨ UA ∩ ∩B - *headsnake, blocked, ispoint.*
- ⇨ ∩A U ∩B - *headsnake, blocked, ispoint, spawn, nextsnake.*
- ⇨ ∩A ∩ ∩B - *headsnake, blocked, ispoint.*
- ⇨ UA - *headsnake, blocked, ispoint, spawn.*
- ⇨ UB - *blocked, headsnake, nextsnake, ispoint, spawn.*
- ⇨ ∩A - *headsnake, blocked, ispoint, spawn.*
- ⇨ ∩B - *blocked, headsnake, nextsnake, ispoint.*

– *Termes* (*sin signo* y *uniendo al principio*):

- ⇨ UA U UB - *height, at, has-block.*
- ⇨ UA ∩ UB - *height, has-block.*
- ⇨ ∩A U UB - *height, at, has-block.*
- ⇨ ∩A ∩ UB - *has-block.*
- ⇨ UA U ∩B - *height, at, has-block.*
- ⇨ UA ∩ ∩B - *has-block.*
- ⇨ ∩A U ∩B - *has-block, at.*

- ⇨ ∩A ∩ ∩B - *has-block.*
- ⇨ UA - *height, at, has-block.*
- ⇨ UB - *height, has-block.*
- ⇨ ∩A - *has-block, at.*
- ⇨ ∩B - *has-block.*

– *Termes* (*sin signo* y *uniendo al final*):

- ⇨ UA U UB - *height, at, has-block.*
- ⇨ UA ∩ UB - *height, has-block.*
- ⇨ ∩A U UB - *height, at, has-block.*
- ⇨ ∩A ∩ UB - *height, has-block.*
- ⇨ UA U ∩B - *height, at, has-block.*
- ⇨ UA ∩ ∩B - *height, has-block.*
- ⇨ ∩A U ∩B - *height, at, has-block.*
- ⇨ ∩A ∩ ∩B - *height, has-block.*
- ⇨ UA - *height, at, has-block.*
- ⇨ UB - *height, has-block.*
- ⇨ ∩A - *height, at, has-block.*
- ⇨ ∩B - *height, has-block.*

– *Termes* (*con signo* y *uniendo al principio*):

- ⇨ UA U UB - *height, at, has-block.*
- ⇨ UA ∩ UB - *height, has-block.*
- ⇨ ∩A U UB - *height, at, has-block.*
- ⇨ ∩A ∩ UB - *has-block.*
- ⇨ UA U ∩B - *height, at, has-block.*
- ⇨ UA ∩ ∩B - *has-block.*
- ⇨ ∩A U ∩B - *has-block, at.*
- ⇨ ∩A ∩ ∩B - *has-block.*
- ⇨ UA - *height, at, has-block.*
- ⇨ UB - *height, has-block.*
- ⇨ ∩A - *has-block, at.*
- ⇨ ∩B - *has-block.*

– *Termes* (*con signo* y *uniendo al final*):

- ⇨ UA U UB - *height, at, has-block.*
- ⇨ UA ∩ UB - *height, has-block.*
- ⇨ ∩A U UB - *height, at, has-block.*
- ⇨ ∩A ∩ UB - *height, has-block.*
- ⇨ UA U ∩B - *height, at, has-block.*
- ⇨ UA ∩ ∩B - *height, has-block.*

- ⇨ ∩A U ∩B - *height, at, has-block.*
- ⇨ ∩A ∩ ∩B - *height, has-block.*
- ⇨ UA - *height, at, has-block.*
- ⇨ UB - *height, has-block.*
- ⇨ ∩A - *height, at, has-block.*
- ⇨ ∩B - *height, has-block.*

– *Hiking* (los conjuntos A y B se componen solo de un grupo de variables, por lo que realizar operaciones internas de union o intersección es irrelevante):

- ⇨ A U B - *at_tent, up, at_person, walked.*
- ⇨ A ∩ B - *at_person, walked.*
- ⇨ A - *at_tent, up, at_person, walked.*
- ⇨ B - *at_person, walked.*

– *Spider* (*uniendo al principio*):

- ⇨ UA U UB - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, collect-card, make-movable, make-unmovable, make-part-of-tableau, in-play, collect-card.*
- ⇨ UA ∩ UB - *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ ∩A U UB - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear, make-movable, make-unmovable, make-part-of-tableau, in-play, part-of-tableau, movable, collect-card.*
- ⇨ ∩A ∩ UB - *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on.*
- ⇨ UA U ∩B - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ UA ∩ ∩B - *on, clear.*
- ⇨ ∩A U ∩B - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear.*
- ⇨ ∩A ∩ ∩B - *on.*
- ⇨ UA - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ UB - *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-*

of-tableau, in-play, part-of-tableau, movable, collect-card, currently-collecting-deck.

- ⇨ ∩A - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on.*
- ⇨ ∩B - *on, clear.*

– *Spider* (*uniendo al final*):

- ⇨ UA U UB - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, collect-card, make-movable, make-unmovable, make-part-of-tableau, in-play, collect-card.*
- ⇨ UA ∩ UB - *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ ∩A U UB - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear, make-movable, make-unmovable, make-part-of-tableau, in-play, part-of-tableau, movable, collect-card.*
- ⇨ ∩A ∩ UB - *currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on.*
- ⇨ UA U ∩B - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ UA ∩ ∩B - *on, clear.*
- ⇨ ∩A U ∩B - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear.*
- ⇨ ∩A ∩ ∩B - *on*.
- ⇨ UA - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*
- ⇨ UB - *on, clear, make-movable, currently-updating-movable, currently-updating-unmovable, make-unmovable, currently-updating-part-of-tableau, make-part-of-tableau, in-play, part-of-tableau, movable, collect-card, currently-collecting-deck.*
- ⇨ ∩A - *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on.*
- ⇨ ∩B - *on, clear.*

– *Tetris*:

- ⇨ UA U UB - *clear, at_square, at_two, at_right_l.*

- ⇨ UA ∩ UB - *clear, at_square, at_two, at_right_l.*
- ⇨ ∩A U UB - *clear, at_square, at_two, at_right_l.*
- ⇨ ∩A ∩ UB - *clear*.
- ⇨ UA U ∩B - *clear, at_square, at_two, at_right_l.*
- ⇨ UA ∩ ∩B - *clear*.
- ⇨ ∩A U ∩B - *clear*.
- ⇨ ∩A ∩ ∩B - *clear*.
- ⇨ UA - *clear, at_square, at_two, at_right_l.*
- ⇨ UB - *clear, at_square, at_two, at_right_l.*
- ⇨ ∩A - *clear*.
- ⇨ ∩B - *clear*.

Tras los anteriores cálculos entre conjuntos, se determina cuáles son las mejores operaciones en cada dominio, es decir, cuáles de las anteriores operaciones se aproximan más al conjunto de predicados considerados como importantes / predicados *relevantes*.

– *Snake*:

- ⇨ Predicados considerados como importantes: *spawn, headsnake, ispoint*.
- ⇨ Mejores operaciones (y más simples):
  - UA: *headsnake, blocked, ispoint, spawn.*
  - ∩A: *headsnake, blocked, ispoint, spawn.*

– *Termes*:

- ⇨ Predicados considerados como importantes: *height, at y has-block.*
- ⇨ Mejores operaciones (y más simples):
  - UA sin signo uniendo al principio: *height, at, has-block.*
  - UA sin signo uniendo al final: *height, at, has-block.*
  - ∩A sin signo uniendo al final: *height, at, has-block.*
  - UA con signo uniendo al principio: *height, at, has-block.*
  - UA con signo uniendo al final: *height, at, has-block.*
  - ∩A con signo uniendo al final: *height, at, has-block.*

– *Hiking*:

- ⇨ Predicados considerados como importantes: *walked, at_person, at_tent, up / down.*
- ⇨ Mejores operaciones (y más simples):
  - A: *at_tent, up, at_person, walked.*

– *Spider*:

- ⇨ Predicados considerados como importantes: *on, clear, currently-collecting-deck, current-deal y currently-dealing.*
- ⇨ Mejores operaciones (y más simples):
  - UA uniendo al principio: *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*
  - UA uniendo al final: *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.*

– *Tetris*:

- ⇨ Predicados considerados como importantes: *clear*.
- ⇨ Mejores operaciones (y más simples):
  - ∩A: *clear*.
  - ∩B: *clear*.

Contemplando las mejores operaciones entre conjuntos se puede llegar a las siguientes conclusiones:

- Las operaciones unitarias sobre el conjunto A aparecen como las mejores en todos los dominios. La cuestión está en decidir si es mejor *UA* o ∩*A*.

- Empíricamente se ha demostrado, al menos en estos dominios, que unir al principio y unir al final proporciona los mismos resultados. Por esa razón se ha elegido unir al principio, pues a fin de cuentas realiza menos operaciones entre conjuntos. Como se dijo anteriormente, en caso de hacer una intersección sobre ese mismo conjunto, implicaría que los predicados *relevantes* deberían aparecer **en el conjunto de todas** las acciones con predicados meta en las acciones. A modo de recordatorio, unir al principio implica crear un único grupo de conjuntos de predicados antes de realizar las operaciones entre conjuntos, con todos los grupos de predicados obtenidos mediante cada una de las metas.

- Empíricamente se ha demostrado, al menos en estos dominios, que tanto tener en cuenta el signo como no tenerlo proporciona los mismos resultados. Dicho esto, dado que al tener en cuenta el signo se opera con menos conjuntos, se ha decidido considerar el signo. A modo de recordatorio, tener en cuenta el signo implica que, al buscar las

metas del problema en los efectos de las acciones, se tendrá en cuenta si este aparece negado (*not*) o afirmado en los problemas.

- Teniendo en cuenta los tres anteriores puntos, se analizarán con detalle las siguientes operaciones:

  - ⇨ *UA con signo y uniendo al principio*.
  - ⇨ *∩A con signo y uniendo al principio*.

- Sobre el conjunto obtenido se realizará una unión con los predicados meta, pues estos son siempre predicados importantes.

- A la hora de elegir entre *UA* y *∩A*, se priorizará el hecho de que haya una variable de más *no relevante* frente al hecho de que falte un predicado *relevante*. Una heurística que ignore algunas variables *relevantes* rechazará, en gran parte, los buenos resultados obtenidos en el estudio dependiente del dominio. Por otra parte, una heurística que tenga en cuenta más variables que las *relevantes* no se contradirá en gran medida con los resultados del estudio dependiente del dominio, y si estas no son realmente beneficiosas, un buen algoritmo de creación de PDBs sabrá prescindir de ellas.

Dicho esto, los resultados son los siguientes:

- ***UA con signo y uniendo al principio***:

  - ⇨ *Snake*:
    - Operación: *(headsnake, blocked, ispoint, spawn)* U *(predicados meta)* - *headsnake, blocked, ispoint, spawn*.
    - Predicados *relevantes*: *headsnake, ispoint, spawn*.
    - Comparación: 1 predicado de más, pero están todos los *relevantes*.
  - ⇨ *Termes*:
    - Operación: *(height, at, has-block)* U *(predicados meta)* - *height, at, has-block.*
    - Predicados *relevantes*: *height, at, has-block.*
    - Comparación: ningún predicado de más, y están todos los *relevantes*.
  - ⇨ *Hiking*:
    - Operación: *(at_tent, up, at_person, walked)* U *(predicados meta)* - *at_tent, up, at_person, walked*.
    - Predicados *relevantes*: *at_tent, up / down, at_person, walked*.
    - Comparación: ningún predicado de más, y están todos los *relevantes*.
  - ⇨ *Spider*:
    - Operación: *(currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card)* U *(predicados meta)* - *currently-dealing, currently-updating-*

movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, current-deal, clear, on, part-of-tableau, movable, in-play, collect-card.
    - Predicados *relevantes*: *currently-dealing, currently-collecting-deck, current-deal, clear, on.*
    - Comparación: 7 predicados de más, pero están todos los *relevantes*.
- ⇨ *Tetris*:
    - Operación: *(clear, at_square, at_two, at_right_l)* U *(predicados meta) - clear, at_square, at_two, at_right_l.*
    - Predicados *relevantes*: *clear*.
    - Comparación: 3 predicados de más, pero están todos los *relevantes*.

- ∩***A con signo y uniendo al principio***:

- ⇨ *Snake*:
    - Operación: *(headsnake, blocked, ispoint, spawn)* U *(predicados meta) - headsnake, blocked, ispoint, spawn*.
    - Predicados *relevantes*: *headsnake, ispoint, spawn*.
    - Comparación: 1 predicado de más, pero están todos los *relevantes*.
- ⇨ *Termes*:
    - Operación: *(has-block, at)* U *(predicados meta) - height, at, has-block*.
    - Predicados *relevantes*: *height, at, has-block*.
    - Comparación: ningún predicado de más, y están todos los *relevantes*.
- ⇨ *Hiking*:
    - Operación: *(at_tent, up, at_person, walked)* U *(predicados meta) - at_tent, up, at_person, walked*.
    - Predicados *relevantes*: *at_tent, up / down, at_person, walked*.
    - Comparación: ningún predicado de más, y están todos los *relevantes*.
- ⇨ *Spider*:
    - Operación*: (currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on)* U *(predicados meta) - currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear.*
    - Predicados *relevantes*: *currently-dealing, currently-collecting-deck, current-deal, clear, on.*
    - Comparación: 3 predicados de más, pero falta uno de los *relevantes* (*current-deal*).
- ⇨ *Tetris*:
    - Operación: *(clear)* U *(predicados meta) - clear*.
    - Predicados *relevantes*: *clear*.
    - Comparación: ningún predicado de más, y están todos los *relevantes*.

Tras este análisis, se puede llegar a la conclusión de que el mejor es ∩*A* teniendo en cuenta el signo y uniéndolos al principio, añadiéndole al final las metas directamente. Son solo 4 predicados de más, y que podrían resultar interesantes, frente a un predicado visto como importante que no aparece. Es posible que este predicado restante se compense con los predicados *no relevantes*. La otra opción, sin embargo, a pesar de incluir todos los predicados relevantes, introducía 11 *no relevantes*, los cuales son muchos. Como trabajo futuro, podría investigarse más a fondo acerca de opción descartada.

A modo de recordatorio, la opción escogida implica identificar aquellas acciones donde haya algún predicado meta en los efectos, teniendo en cuenta si está, o no, con *not* (dependiendo de cómo aparezca en las metas del problema). Por cada acción identificada, se crea un conjunto con los predicados de sus precondiciones. Posteriormente, se recopilan solo aquellos predicados que aparezcan en todos esos conjuntos, añadiendo también los predicados meta.

Llegados a este punto, habría que establecer cómo pasar de predicados *relevantes* a variables *relevantes*. Dado que una variable está compuesta de varias instanciaciones de predicados, se pueden establecer dos alternativas:

- La primera de ellas es coger aquellas variables que estén compuestas de al menos una instancia de un predicado *relevante*. Esta es la opción más simple y la que finalmente se ha elegido.

- La segunda es realizar un estudio de aquellas instancias de predicados que podrían resultar interesantes, mediante el análisis del tipo de los objetos que acompañan a los predicados *relevantes*. Esto filtraría bastantes variables que no proporcionarían buenos resultados. La razón por la cual no se ha decidido tomar esta alternativa es que complementaría a la decisión de recopilar predicados *relevantes*, no solo teniendo en cuenta el signo, sino también el tipo de objetos a los que acompaña en los efectos de las acciones y en la meta. Un trabajo futuro denso podría consistir en contrastar los resultados obtenidos en este proyecto con los resultados obtenidos mediante otro estudio paralelo que considerase también el tipo de los objetos, tanto al recopilar los predicados *relevantes*, como al seleccionar aquellas variables *relevantes* dentro del *output.sas*. De todos modos, esta elección tampoco afecta mucho en este proyecto, ya que solo en el dominio *Spider* hay predicados con el mismo nombre pero con distinto tipo de objetos; y muchas variables contenían, dentro de *output.sas*, instanciaciones de estos predicados junto a los posibles tipos de objetos.

El siguiente paso es orientar la investigación a partir de ahora, considerando que ya se tienen las variables *relevantes*. Existen diversas alternativas. A continuación, se exponen tres de ellas y se indica cuál se ha seleccionado para llevar a cabo:

- **Crear un algoritmo que construya PDBs acertadas a partir de esas variables *relevantes***. A simple vista podría resultar la mejor alternativa, pero tras observar cómo se agrupan las variables en los estudios dependientes del dominio no parece que exista una correlación trivial entre las características del dominio y la combinación de variables (conjuntos de PDBs) obtenidas como importantes. Por ejemplo, en *Termes* hay una cuadrícula con posiciones, al igual que en *Tetris*. En Termes se vio que era una buena idea tener una PDB por cada combinatoria que existe entre los vecinos de las alturas variables. Pero en Tetris se tuvo en cuenta las casillas por fila, ya que en esta versión del juego el objetivo era dejar como *clear* las casillas superiores. Un dominio tenía como meta la altura y el otro *clear*, estos acompañados de tipos de objetos distintos. ¿Cómo se podría extraer del dominio qué variables son importantes mediante el estudio de todas las características de este? ¿Cómo llegar a la conclusión, mediante un algoritmo programado, de que importan más los clear de una fila o la combinación de las posiciones vecinas a las alturas variables, teniendo en cuenta cosas como, por ejemplo, la connectividad entre casillas, la movilidad, etc.? ¿Cómo generalizar todo esto para cada dominio?

En resumen, cada dominio y problema tiene unas características concretas, por cómo son las acciones, cómo funciona todo, cuáles son los tipos presentes en el problema, etc.; que implica que los conjuntos de PDBs funcionen mejor ordenando unos patrones de una forma o de otra, construyendo un número de PDBs en concreto, cada una de ellas con unas variables en cuestión ... Esto podría ser un trabajo de investigación arduo y complejo a la altura de otro Trabajo Fin de Grado que lo abarcase.

Como posibles ideas, un algoritmo que puede hacer esto podría construirse mediante genéticos. *iPDB*, en el fondo, tiene incorporado un algoritmo de escalada que construye PDBs, por lo que también podría considerarse. Otra posible idea sería realizar un gran estudio mediante la observación del dominio y de las PDBs que se obtienen mediante diversos algoritmos, con el fin de buscar una correlación e implementar un programa que construyese el conjunto de patrones.

- **Modificar *iPDB* para que solo considere variables *relevantes* al ir construyendo los patrones.** iPDB es un algoritmo que construye PDBs a partir de un conjunto de variables, exactamente lo que se buscaba en la anterior alternativa descartada. Considerar solo aquellas variables *relevantes* reduciría el tiempo de iPDB. Por otra parte, también puede suceder que resuelva otros problemas en menos tiempo, o que sea capaz de resolver algunos que abortaban su ejecución en el proceso de construcción de PDBs debido al tiempo y la memoria empleada. También es posible que se obtengan PDBs más enfocadas, usando menos memoria, encontrando soluciones que el *iPDB* sin modificar no podría. Sin embargo, confiar en este método implica una desconfianza en *iPDB*. Reducir a 0 el *improvement* de variables a priori *no relevantes*, pero prometedoras. De esa forma, sería posible el hecho de que empezase a resolver menos problemas, pues se reduce el tamaño de variables que analizar. En algunos problemas quizás sean

importantes variables que no se tienen en cuenta en esta modificación de *iPDB*. Al fin y al cabo, *iPDB* no generaliza, se adapta al problema y dominio en cuestión, por lo que ser tan radical podría llegar a mejorar los problemas mucho, poco o nada. Por todas estas razones, se ha terminado descartando para el proyecto, pero podría ser una alternativa a estudiar en futuros trabajos. En caso de tomar esta alternativa, para favorecer la ejecución, estaría bien bajar el *improvement*, refinarlo, al igual que aumentar el tamaño de las PDBs. En un caso opcional, redudirle el *max_time* si aún se quedase atascado.

- **Modificar *iPDB*, aumentando el *improvement* de las variables *relevantes* o bajándoselo a las *no relevantes* para forzar un poco su elección**. Es difícil estimar si el tiempo de las ejecuciones en este caso será menor o mayor, pero si las variables *relevantes* son adecuadas, es posible que se refinen las PDBs creadas. En este caso, si una variable *no relevante* es potencialmente buena, *iPDB* la considerará. De igual forma que en la segunda alternativa, se podría probar reduciéndole el *max_time*, variando el *improvement*, ampliando la memoria, etc. Las alternativas son subir improvement de las *relevantes*; bajárselas a las *no relevantes*; y hacer las dos cosas a la vez. A continuación, se analiza cada alternativa:

  - ⇨ **Bajar el *improvement***: Si una variable *no relevante* es aparentemente buena por *iPDB* (obteniendo un *improvement* elevado obtenido por mediante sus *samples*), no resultaría una buena idea menospreciarla, ya que para ese dominio y ese problema en cuestión probablemente tenerla en cuenta origine buenos resultados. *iPDB* no sobrevalora variables potencialmente buenas en una PDB. Sin embargo, considerar esta alternativa y llevarla a cabo en un trabajo futuro no significa que sea una idea descabellada, aunque no se va a abordar en este proyecto.
  - ⇨ **Subir el *improvement***. Es más lógico pensar que *iPDB* esté menospreciando más de la cuenta una variable potencialmente buena. Por esa razón, se considera mejor alternativa aumentar el *improvement* cuando se analicen variables *relevantes.* De esta forma, frente a dos nuevas PDBs, una con *n* variables *relevantes* y la otra con *n-1 relevantes* y una *no relevante*, si el *improvement* es parecido o ligeramente superior en una que en otra, el algoritmo se decantará por la PDB que incluya las *n* variables *relevantes*. Este incentivo al *improvement* tiene más sentido que sea multiplicativo, ya que el número de *samples* de *iPDB* puede variar de una ejecución a otra. De esta forma el incentivo es equitativo.
    El cálculo del factor es simple: comenzando en valor uno, se incrementará una unidad por cada variable *relevante* de la PDB. De esta forma, frente a dos PDBs, una con pocas variables *no relevantes* y otra con muchas variables *no relevantes*, para que este último sea elegido su *improvement* deberá sobrepasar bastante el valor del *improvement* del patrón con pocas *relevantes*, incentivando aún más estas variables. Cuanto mayor sea este factor, con mayor probabilidad se desecharán PDBs que añadan variables *no relevantes*. De todos modos, plantear

diferentes formas de incentivar las variables *relevantes* podría ser objeto de futuras investigaciones comparativas.
Por otra parte, se ha considerado una mejor alternativa incentivar estas PDBs siempre y cuando su *improvement* supere el *min_improvement*, para darle sentido al parámetro. Es decir, si a una PDB se le introdujera una variable *relevante,* pero tuviese improvement 9 (siendo el mínimo 10), no se incentivará multiplicando el 9 por 2.

- ⇨ **Subir el *improvement* de las *relevantes* y bajar el improvement de las *no relevantes*.** Dado que se ha descartado el hecho de bajar el improvement, también se descarta esta alternativa, aunque, repito, no significa que abordarla en un trabajo sea una idea descabellada.

Tras seleccionar el método de trabajo, se van a establecer las siguientes premisas de cara a la implementación del algoritmo y la modificación de *iPDB*:

- A modo de resumen, el algoritmo que se va a implementar hará lo siguiente:

  - ⇨ Identificar aquellas acciones donde haya algún predicado meta en los efectos, teniendo en cuenta si está, o no, con *not* (dependiendo de cómo aparezca en las metas del problema). Por cada acción identificada, se crea un conjunto con los predicados de sus precondiciones. Posteriormente, se recopilan solo aquellos predicados que aparezcan en todos esos conjuntos, añadiendo también los predicados meta. Estos predicados se denominan predicados *relevantes*.
  - ⇨ Crear una lista con todas las variables *relevantes*. Estas se extraerán del archivo *output.sas*. Estas variables son las que contengan alguna instanciación de un predicado *relevante*.
  - ⇨ Llevar esta lista a *iPDB* modificado.

- A modo de resumen, *iPDB* modificado hará lo siguiente:

  - ⇨ Ejecutar tal y como hace *iPDB*, pero multiplicando el *improvement* (siempre y cuando supere un valor mínimo) por un factor. Este factor se obtiene contando el número de variables *relevantes* de la PDB en cuestión. De esta forma, se incentivará la elección de PDBs con variables de esta índole.

- En el análisis dependiente del dominio se justificó cuáles eran los parámetros en *iPDB* que se iban a introducir. Se va a conservar el mismo criterio. Se llegó a la conclusión de que la mejor alternativa era ejecutar *iPDB modificado* sin variar ningún parámetro en los dominios *Snake, Termes, Hiking* y *Spider*, y acortar el *max_time* a un minuto en el dominio *Tetris*, tal y como se hizo en *iPDB*. De esta forma se podrán realizar mejores comparaciones entre los dos algoritmos *iPDB*, pues la idea principal es observar si esta modificación mejora el algoritmo original.

- Las características en común que existen entre el algoritmo *iPDB* modificado, junto al algoritmo de extracción de variables relevantes implementado, y los cinco algoritmos dependientes del dominio anteriormente codificados son las siguientes:

  - ⇨ Ambos incluyen todas las variables meta en sus PDBs, aunque sea mediante PDBs con esa única variable.
  - ⇨ El 80% de los algoritmos dependientes del dominio funcionaban mejor mediante *Canonical*, e *iPDB* funciona mediante *Canonical*. Dicho esto, el orden de las PDBs no es relevante.
  - ⇨ Todos esos algoritmos tienen en cuenta las variables *relevantes*, que coinciden en gran parte unas con otras. Todas tienen predicados que aparecen en las precondiciones de las acciones que causan un cambio en los predicados meta, al menos en los dominos estudiados.
  - ⇨ Todos estos algoritmos tienen su origen en un estudio de *iPDB*, su funcionamiento, sus ejecuciones y resultados.
  - ⇨ Actúan sobre los mismos 5 dominios, acompañados de los mismos problemas. Además, todas las metas de todos los problemas de un mismo dominio son del mismo estilo, es decir, se componen de los mismos predicados.

- Las variables *relevantes* por el algoritmo independiente del dominio pero no por los algoritmos dependientes del dominio, si no son potencialmente buenas, no serán escogidos de todos modos por *iPDB* modificado. Aunque se multiplique su *improvement* por dos, no llegarán a superar a la PDB con el mayor *improvement* de esa iteración.

- En las pruebas del algoritmo bastará con elegir un problema por dominio, ya que con todos los problemas de un dominio en concreto el algoritmo implementado actúa de forma similar, pues se parsea el propio dominio. Respecto a la prueba de *iPDB* modificado, tan solo bastará con observar si duplica el *improvement* de las variables *relevantes* obtenidas en el otro algoritmo implementado. Esta última prueba se realizará simplemente observando las trazas de ejecución y verificando su correcto funcionamiento en el documento. Incluir esta prueba de forma íntegra en el apartado 4.7.3 Pruebas del algoritmo puede llegar a ser extenso e irrelevante, aunque se haga solo sobre un problema de cada dominio. Este consistiría en recopilar todas aquellas PDBs donde se añada una variable que incentive el *improvement* y compararlas con las variables *relevantes*. También habría que recopilar todas aquellas PDBs donde se añada una variable que no incentive el *improvement* y compararlas con las variables *no relevantes*. Dado que en usa sola ejecución pueden presentarse miles de PDBs, mostrar esto resulta inviable.

- Con el fin de comprobar si esta heurística mejora o no los resultados, se comparará con *iPDB* sin modificar y con el mejor algoritmo obtenido en el estudio dependiente del dominio.

4.7.2 Implementación y ejecución del algoritmo y/o modificación y ejecución de algoritmos independientes del dominio

En este punto se implementará el algoritmo de acorde a las conclusiones extraídas en el anterior punto. En esas conclusiones también se decidió modificar *iPDB*. El script implementado se comenta en el punto 3.3.8 *Script* del algoritmo independiente del dominio y el algoritmo *iPDB* modificado en el punto 3.3.9 Modificaciones en *iPDB*. Los resultados, dado que se van a comparar posteriormente con iPDB sin modificar y con el mejor algoritmo obtenido en el estudio dependiente del dominio, se muestran en conjunto (*tablas 160 – 174 y 213 – 409*):

Tiempo total (s)	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	897.092	5.80153393	200.58852195358276
P02	279.163	9.94346408	270.16790111160276
P03	361.447	15.90446882	351.4272340373993
P04	84.5717	0.23493203	366.1114518699646
P05	-	0.66840707	292.0023769435883
P06	446.45	13.21105392	447.8545679283142
P07	513.355	54.68040521	517.127660987854
P08	-	659.32719686	-
P09	912.248	0.48888902	406.33500299072267
P10	430.939	7.01122390	284.3349829940796
P11	641.723	15.66071493	644.4128170833587
P12	666.832	51.33190990	667.6770430583954
P13	-	-	-

P14	-	-	-
P15	954.978	2.79343696	376.0553479423523
P16	709.584	29.22603607	713.4984349422455
P17	-	634.81734115	-
P18	-	-	-
P19	-	-	-
P20	1173.33	10.23564088	772.955080953598

Tabla 160. Tiempo total – Snake – DI

Tiempo total (s)	*iPDB*	*AlgTermes (Canonical PDB)*	*iPDB modificado*
P01	3.60109	0.11834303	3.4059978691864012
P02	3.58253	0.62873187	3.4644889012145996
P03	4.85755	6.09257109	4.940265952758789
P04	7.34706	5.56850395	7.196026974334717
P05	271.911	-	271.8283199996948
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	5.7526	1.23375510	6.081440991287232

P12	3.60016	1.28281204	3.7337870392608643
P13	58.1779	53.44804313	58.321744946289066
P14	11.7071	35.62901007	11.823701928329468
P15	-	-	-
P16	-	-	-
P17	81.8209	100.35224798	82.09607600097657
P18	15.6232	14.63828609	15.611186093139649
P19	27.0189	58.01016885	27.33332597236633
P20	126.88	73.85213813	126.90598094749451

Tabla 161. Tiempo total – Termes – DI

Tiempo total (s)	*iPDB*	*AlgHiking (Canonical PDB)*	*iPDB modificado*
P01	0.032566	0.00318769	0.03474373495788574
P02	0.0783462	0.00661312	0.07385205054016113
P03	0.127391	0.03501052	0.09031453038024902
P04	0.272573	0.18526599	0.2793440118713379
P05	0.518506	0.36832912	0.5505571138763428
P06	0.209041	0.03768729	0.21956096047973633
P07	3.69358	0.19507193	3.951463186645508
P08	52.8204	1.49180899	53.73265293731689
P09	450.684	11.31199909	457.8032989883423

P10	-	53.02414192	-
P11	-	243.73920004	-
P12	14.7052	0.59621516	14.731462091064452
P13	269.414	6.17371200	274.6540928707123
P14	-	42.05324300	-
P15	-	274.48803722	-
P16	0.877289	0.06554133	0.7619059916992188
P17	48.6175	1.23725816	47.05319116592407
P18	-	14.91013789	-
P19	-	146.56743888	-
P20	-	-	-

Tabla 162. Tiempo total – Hiking – DI

Tiempo total (s)	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	3.89944	0.13522190	3.8798289346313477
P02	8.88602	0.24146891	8.814547993011475
P03	16.5604	1.04915595	16.58043108100891
P04	45.8777	2.19571807	15.059381076049805
P05	52.7484	7.82037395	52.79493982887268
P06	-	-	-
P07	3.16054	0.20975509	3.1757199687194824

P08	4.57249	0.33268491	4.633323965530396
P09	14.9599	1.28919501	14.948388869476318
P10	132.884	3.30433917	26.656401070022582
P11	291.744	283.11515784	299.2904289627075
P12	-	-	-
P13	919.798	-	-
P14	30.1373	0.14392103	2.925075022277832
P15	5.04144	0.25848996	5.133413017883301
P16	15.4625	3.26207011	15.736908097076416
P17	28.105	14.22202295	27.85382001991272
P18	427.141	-	428.7959159145355
P19	-	-	-
P20	-	-	-

Tabla 163. Tiempo total – Spider – DI

Tiempo total (s)	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*
P01	116.675	8.16044188	116.9020539264679
P02	128.187	10.26613210	87.70037000350952
P03	-	-	-
P04	77.5581	0.44831291	73.30335084571838

P05	63.6368	9.86774204	63.7614490032196
P06	-	24.22178288	-
P07	-	-	-
P08	63.9123	0.41998802	107.79069302177429
P09	99.6254	8.36833290	99.86123203735352
P10	125.007	19.03626089	125.56821297264099
P11	-	-	-
P12	65.8284	8.09096503	115.03603100967408
P13	-	212.04270498	-
P14	-	-	-
P15	88.3681	8.07321586	88.62776788406372
P16	-	-	-
P17	-	288.63949498	-

Tabla 164. Tiempo total – Tetris – DI

Número nodos expandidos	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	44	50	44
P02	5188	6562	5188
P03	1264851	400056	1264851
P04	13	13	13

P05	-	18	18
P06	333	1123	333
P07	15710618	2757587	15710618
P08	-	44679351	-
P09	21	21	21
P10	43	62	43
P11	371	395	371
P12	9095686	1876776	9095686
P13	-	-	-
P14	-	-	-
P15	44	46	46
P16	248924	52478	248924
P17	-	33907904	-
P18	-	-	-
P19	-	-	-
P20	31	31	31

Tabla 165. Número nodos expandidos – Snake – DI

Número nodos expandidos	*iPDB*	*AlgTermes (Canonical PDB)*	*iPDB modificado*
P01	4776	25356	4776
P02	17874	153247	17874
P03	145662	1630019	145662
P04	748085	1449714	748122
P05	74493127	-	74493127
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	219376	137102	694212
P12	52374	71225	52374
P13	14958320	13849206	14958320
P14	1953682	9693366	1953682
P15	-	-	-
P16	-	-	-
P17	22443585	22629895	22443585
P18	3403414	2316184	3403414

P19	6027955	3832006	6027955
P20	34655820	17442585	34655820

Tabla 166. Número nodos expandidos – Termes – DI

Número nodos expandidos	***iPDB***	***AlgHiking (Canonical PDB)***	***iPDB modificado***
P01	26	26	26
P02	301	301	301
P03	2140	2140	2140
P04	30812	30812	30812
P05	77262	77262	77262
P06	10398	495	10398
P07	477813	9903	477813
P08	6901037	97234	6901037
P09	59226603	875690	59226603
P10	-	4335060	-
P11	-	22704671	-
P12	1548040	25862	1548040
P13	28004230	318545	28004230
P14	-	2432003	-
P15	-	17011918	-

P16	48402	2119	48402
P17	4107620	46955	4107620
P18	-	634659	-
P19	-	6751727	-
P20	-	-	-

Tabla 167. Número nodos expandidos – Hiking – DI

Número nodos expandidos	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	2694	2808	2694
P02	4854	4815	4854
P03	32275	39925	32275
P04	29191	86492	86747
P05	288174	248806	288174
P06	-	-	-
P07	2578	3040	2578
P08	6578	7936	6578
P09	47373	51900	47373
P10	23897	158754	134254
P11	11896317	14289345	11896317
P12	-	-	-

P13	15093157	-	-
P14	892	4371	3117
P15	561	645	561
P16	137211	218488	137211
P17	612546	933749	612546
P18	17748489	-	17748489
P19	-	-	-
P20	-	-	-

Tabla 168. Número nodos expandidos – Spider – DI

Número nodos expandidos	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*
P01	169372	1501	169372
P02	127104	4431	93058
P03	-	-	-
P04	17	7	17
P05	55472	1218	55472
P06	-	437305	-
P07	-	-	-
P08	10	10	10
P09	29690	17	29690

P10	1829129	90540	1829129
P11	-	-	-
P12	285624	1642	160605
P13	-	5143380	-
P14	-	-	-
P15	1698	102	1698
P16	-	-	-
P17	-	5212715	-

Tabla 169. Número nodos expandidos – Tetris – DI

Coste de la solución	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	24	24	24
P02	32	32	32
P03	43	43	43
P04	12	12	12
P05	-	17	17
P06	31	31	31
P07	48	48	48
P08	-	58	-
P09	20	20	20
P10	27	27	27

P11	36	36	36
P12	47	47	47
P13	-	-	-
P14	-	-	-
P15	25	25	25
P16	42	42	42
P17	-	62	-
P18	-	-	-
P19	-	-	-
P20	30	30	30

Tabla 170. Coste de la solución – Snake – DI

Coste de la solución	*iPDB*	*AlgTermes (Canonical PDB)*	*iPDB modificado*
P01	36	36	36
P02	54	54	54
P03	68	68	68
P04	80	80	80
P05	132	-	132
P06	-	-	-
P07	-	-	-

P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	66	66	66
P12	46	46	46
P13	92	92	92
P14	104	104	104
P15	-	-	-
P16	-	-	-
P17	116	116	116
P18	76	76	76
P19	94	94	94
P20	106	106	106

Tabla 171. Coste de la solución – Termes – DI

Coste de la solución	*iPDB*	*AlgHiking (Canonical PDB)*	*iPDB modificado*
P01	11	11	11
P02	17	17	17
P03	25	25	25
P04	38	38	38
P05	45	45	45

P06	10	10	10
P07	16	16	16
P08	22	22	22
P09	30	30	30
P10	-	34	-
P11	-	42	-
P12	17	17	17
P13	24	24	24
P14	-	28	-
P15	-	35	-
P16	10	10	10
P17	17	17	17
P18	-	23	-
P19	-	30	-
P20	-	-	-

Tabla 172. Coste de la solución – Hiking – DI

Coste de la solución	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	16	16	16
P02	23	23	23
P03	22	22	22

P04	31	31	31
P05	50	50	50
P06	-	-	-
P07	16	16	16
P08	18	18	18
P09	25	25	25
P10	27	27	27
P11	29	29	29
P12	-	-	-
P13	39	-	-
P14	10	10	10
P15	17	17	17
P16	17	17	17
P17	23	23	23
P18	27	-	27
P19	-	-	-
P20	-	-	-

Tabla 173. Coste de la solución – Spider – DI

Coste de la solución	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*

P01	30	30	30
P02	36	36	36
P03	-	-	-
P04	10	10	10
P05	30	30	30
P06	-	39	-
P07	-	-	-
P08	11	11	11
P09	21	21	21
P10	48	48	48
P11	-	-	-
P12	27	27	27
P13	-	56	-
P14	-	-	-
P15	19	19	19
P16	-	-	-
P17	-	66	-

Tabla 174. Coste de la solución – Tetris – DI

- El resto de información se puede observar en las tablas que se encuentran en la Sección 9 Anexo.

4.7.3 Pruebas del algoritmo

Se comparará la salida esperada con la obtenida, y si coincidiesen, se verificará su correcto funcionamiento. A su vez, dado que se ajusta a los requisitos de usuario, también se validará. Dicho esto, se van a observar los problemas número 1 de cada dominio, pues son resueltos por el algoritmo.

El algoritmo implementado sobre el problema 1 del dominio *Snake* muestra las siguientes variables como *relevantes*:

- Predicados *relevantes* esperados por el algoritmo: h*eadsnake, blocked, ispoint, spawn.*

- Variables *relevantes* esperadas:

  - ⇨ *headsnake*: 107.
  - ⇨ *blocked*: 80 – 87 y 89 – 105.
  - ⇨ *ispoint*: 108 – 122.
  - ⇨ *spawn*: 88.

- Variables *relevantes* obtenidas: 80 – 105 y 107 – 122.

Por otra parte, *iPDB* modificado incentiva el *improvement* (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1).

El algoritmo implementado sobre el problema 1 del dominio *Termes* muestra las siguientes variables como *relevantes*:

- Predicados *relevantes* esperados por el algoritmo: *height*, *at*, *has-block.*

- Variables *relevantes* esperadas:

  - ⇨ *height*: 1 – 11.
  - ⇨ *at*: 0.
  - ⇨ *has-block*: 12

- Variables *relevantes* obtenidas: 0 – 11.

Por otra parte, *iPDB* modificado incentiva el *improvement* (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1).

El algoritmo implementado sobre el problema 1 del dominio *Hiking* muestra las siguientes variables como *relevantes*:

- Predicados *relevantes* esperados por el algoritmo: *at_tent, up, at_person, walked*.

- Variables *relevantes* esperadas:

- ⇨ *at_tent*: 3.
- ⇨ *up*: 0.
- ⇨ *at_person*: 4, 5.
- ⇨ *walked*: 6.

- Variables *relevantes* obtenidas: 0 y 3 – 6.

Por otra parte, *iPDB* modificado incentiva el *improvement* (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1).

El algoritmo implementado sobre el problema 1 del dominio *Spider* muestra las siguientes variables como *relevantes*:

- Predicados *relevantes* esperados por el algoritmo: *currently-dealing, currently-updating-movable, currently-updating-unmovable, currently-updating-part-of-tableau, currently-collecting-deck, on, clear*.

- Variables *relevantes* esperadas:

- ⇨ *currently-dealing*: 1.
- ⇨ *currently-updating-movable*: 152.
- ⇨ *currently-updating-unmovable*: 150.
- ⇨ *currently-updating-part-of-tableau*: 151.
- ⇨ *currently-collecting-deck*: 98.
- ⇨ *on*: 153 – 155 y 158 – 166.
- ⇨ *clear*: 104, 128, 129, 131, 142 – 149, 156, 157 y 167 – 169.

- Variables *relevantes* obtenidas: 1, 98, 104, 128, 129, 131 y 142 – 169.

Por otra parte, *iPDB* modificado incentiva el *improvement* (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1).

El algoritmo implementado sobre el problema 1 del dominio *Tetris* muestra las siguientes variables como *relevantes*:

- Predicados *relevantes* esperados por el algoritmo: *clear*.

- Variables *relevantes* esperadas:

  ⇨ *clear*: 407 – 430.

- Variables *relevantes* obtenidas: 407 – 430.

Por otra parte, *iPDB* modificado incentiva el *improvement* (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1).

Se puede observar claramente cómo las variables que identifica el algoritmo coinciden con las variables *relevantes*. A su vez, se indica en cada una de las pruebas que *iPDB* modificado incentiva el improvement (siempre y cuando alcance el valor mínimo de 10) de aquellas PDBs con variables *relevantes*, y tantas veces como variables de estas características haya (sumándole 1). Por ejemplo, si el *improvement* fuese de 11 en un patrón y hubiese 1 variable relevante en dicha PDB, el *improvement* resultante sería de 11*(1+1) = 22.

Como se puede comprobar, los algoritmos funcionan correctamente. Quedan, por tanto, verificados y validados. Dadas las características de ambos programas, un análisis más exhaustivo relacionado con una metodología más dura no tendría demasiado sentido.

4.7.4 Evaluación del algoritmo

En este apartado se comparan los resultados del algoritmo implementado con *iPDB* y el mejor algoritmo implementado anteriormente en cada dominio. Estos algoritmos son *AlgSnake* en *Canonical*, *AlgTermes* en *Canonical*, *AlgHiking* en *Canonical*, *AlgSpider* en *Zero-One* y *AlgTetris* en *Canonical*. Además, dado que los resultados de cada dominio son ligeramente distintos, se analizarán por separado:

4.7.4.1. *Snake*

Se puede apreciar (*tablas 160, 165, 170, 213, 218, 223 y 228*) cómo *iPDB* modificado emplea menos tiempo para creación de patrones que *iPDB* en mucho de los problemas. Sin embargo, en el problema 4 las tornas se invierten. De todos modos, los tiempos suelen ser parecidos, quizás ligeramente mayores en el modificado, dado que tienen que ejecutar más código por iteración. Estos tiempos no llegan a ser tan pequeños como los de *AlgSnake*. Y, como es de esperar, el tiempo de poda y creación de conjuntos aditivos es despreciable en cualquiera de los tres algoritmos.

Respecto al tiempo de búsqueda y número de nodos expandidos, no se aprecia ninguna mejora significante en el algoritmo modificado con respecto a *iPDB*. El análisis del tiempo total, por tanto, es equivalente al del tiempo de generación de PDBs.

La memoria total utilizada disminuye en aquellos problemas donde el tiempo de generación de patrones en *iPDB* modificado destaca frente al algoritmo sin modificar, y aumenta cuando *iPDB* tarda menos en construir las PDBs. En caso contrario, cuando ambos tiempos son similares, la memoria utilizada es la misma o muy similar. A pesar de ello, no suelen ser tan pequeños como los de *AlgSnake*. Y, finalmente, los problemas resueltos por el algoritmo modificado son los mismos que en el algoritmo sin modificar, a excepción de aquel problema donde *iPDB* se quedaba bloqueado en la creación de PDBs. En cuestión de coste de la solución, tal y como se puede intuir, todos son iguales.

4.7.4.2. *Termes*

Se puede apreciar (*tablas 161, 166, 171, 214, 219, 224 y 229*) cómo *iPDB* modificado emplea prácticamente el mismo tiempo para creación de PDBs que *iPDB*. En algunas ocasiones es ligeramente menor y otras veces ligeramente mayor porque este primero tiene que ejecutar más código por iteración. Y, como es de esperar, el tiempo de poda y creación de conjuntos aditivos es despreciable en cualquiera de los tres algoritmos.

Respecto al tiempo de búsqueda y número de nodos expandidos, no se aprecia ninguna mejora significativa en el algoritmo modificado con respecto a *iPDB*, a excepción del problema 11, donde se aprecian valores muchísimo más grandes en el modificado. El análisis del tiempo total, por tanto, resulta muy similar en los dos algoritmos.

La memoria total utilizada también es prácticamente la misma en estos dos algoritmos, exceptuando el problema 11, donde se puede ver cómo es mayor en *iPDB*. Esto significaría que la heurística obtenida en el problema 11 es más informada, pero requiere de más memoria. Debido a que todas las variables en este problema son *relevantes*, este inventivo se dará, por tanto, en aquellas PDBs con más variables, lo cual podría suponer una mejora. Sin embargo, esto no es así en todos los casos, ya que en muchas ocasiones son mejores pocas variables combinadas que muchas con poca relación entre ellas. En el problema 11 el hecho de conseguir PDBs más grandes eclipsó las PDBs ligeramente más pequeñas, pero mejor informadas. Sin embargo, en *AlgTermes* fue distinto, empleando menos memoria, incluso, que *iPDB* modificado. Y, finalmente, los problemas resueltos por el algoritmo modificado son los mismos que en el algoritmo sin modificar, al igual que sus costes.

4.7.4.3. *Hiking*

En *Hiking* (*tablas 162, 167, 172, 215, 220, 225 y 230*), los tiempos de generación de PDBs, el de poda y creación de conjuntos aditivos son despreciables en cualquiera de los tres algoritmos. El número de nodos expandidos, el número de problemas resueltos y el valor del coste de la solución son los mismos en los dos algoritmos *iPDB*; y la memoria, el tiempo de búsqueda y el tiempo total muy similares.

4.7.4.4. *Spider*

Se puede apreciar (*tablas 163, 168, 173, 216, 221, 226 y 231*) cómo *iPDB* modificado emplea prácticamente el mismo tiempo para creación de PDBs que *iPDB*. Sin embargo, en los problemas 4, 10, 13 y 14 es mucho menor en el modificado, pero no llegando a alcanzar al de *AlgSpider*. Como es de esperar, el tiempo de poda y creación de conjuntos aditivos es despreciable en cualquiera de los tres algoritmos.

Respecto a la memoria, número de nodos expandidos y el tiempo de búsqueda, *iPDB* muestra unos valores más pequeños que su versión modificada tan solo en los problemas 4, 10, 13 y 14; pero estos son prácticamente los mismos en el resto de los problemas. Respecto al tiempo total, ambos también son similares, a excepción de los problemas anteriormente nombrados, en los que se muestran valores más pequeños en el modificado, ya que el tiempo de búsqueda ahorrado no contrarresta el tiempo empleado en la generación de PDBs en *iPDB*. Sin embargo, en el problema 13 no sucede esto, pues la versión modificada no es capaz de resolverlo.

Por otro lado, *AlgSpider* en estos problemas muestra unos valores de tiempo de búsqueda ligeramente mayores que en *iPDB* modificado, pero respecto al tiempo total, las tornas se invierten. La memoria total utilizada en estos problemas es muy variante en *AlgSpider*, es decir, en los problemas 4 y 14 tiene un valor intermedio entre *iPDB* e iPDB modificado, pero en el problema 10 supera a la versión modificada. Sin embargo, respecto al número de nodos expandidos en los problemas 10 y 14, este es mayor en *AlgSpider*, pero tan solo ligeramente menor en este último con respecto a *iPDB* modificado en el problema 4.

Es importante destacar que los problemas 4, 10, 13 y 14 son aquellos en los que las variables que conforman los patrones de iPDB son aquellas poco usuales, como *collect-card*. Tiene sentido que iPDB modificado no supiera tratar demasiado bien estos problemas.

Para terminar, *iPDB* modificado resuelve los mismos problemas que *AlgSpider* más el problema 18, e *iPDB*, a su vez, resuelve todos los de su versión modificada más el problema 13. Todos los planes tienen el mismo coste.

4.7.4.5. *Tetris*

Respecto al *Tetris* (*tablas 164, 169, 174, 217, 222, 227 y 232*), el tiempo de generación de patrones y la memoria total utilizada en *iPDB* es similar respecto a su versión modificada, pero varía en algunos problemas, donde es mayor en uno que en otro. De todos modos, no llega ninguno de los dos a valores tan bajos de *AlgTetris*. Como es de esperar, el tiempo de poda y creación de conjuntos aditivos es despreciable en cualquiera de los tres algoritmos.

Por otra parte, la heurística es más informada en *iPDB* modificado, pero no al nivel de *AlgTetris*, lo que conlleva que el tiempo de búsqueda no sea tan diferenciador en los dos algoritmos *iPDB*. El tiempo total queda, por tanto, determinado por el tiempo de creación de patrones. Además, el número de problemas resueltos por *iPDB* e *iPDB* modificado es el mismo. Finalmente, el coste de la solución de los algoritmos es el mismo.

4.7.5 Extracción de conclusiones

En primer lugar, los algoritmos que crean una heurística dependiente del dominio son mejores que *iPDB* modificado, pues se asemeja a *iPDB*. De forma empírica se ha demostrado que no sería una buena alternativa anteponer, en términos generales, este algoritmo modificado frente a cualquiera de los otros 5 implementados. Por otra parte, en caso de no realizar el proceso de investigación para implementar el algoritmo dependiente del dominio, la utilización de *iPDB* o de su versión modificada radica en las características del dominio y del problema a resolver en sí, no en la potencia de dichos algoritmos. Básicamente, si la aparición de variables *relevantes* en mayor medida favorece a la resolución del dominio y del problema, este algoritmo modificado puede ser la mejor alternativa. Elegir uno de estos dos como mejor es imposible. Es por eso por lo que una buena alternativa sería ejecutar el algoritmo dependiente del dominio, si no resuelve el problema ejecutar uno de los *iPDB*, si sigue sin resolverlo ejecutar el otro, y en última instancia ejecutar los algoritmos *combo*.

SECCIÓN 5. MARCO REGULADOR

En este punto se abordará el marco regulador aplicado al proyecto en tres puntos: un análisis de la legislación aplicable sobre la implementación descrita en el trabajo, estándares técnicos y un estudio de las cuestiones relacionadas con la propiedad intelectual de la idea.

5.1 Análisis de la legislación aplicable

En este subpunto se comentarán aspectos relacionados con los riesgos, las responsabilidades profesionales, privacidad y seguridad, etc. Es decir, se analizarán cuestiones relacionadas con la legislación que se aplica al presente proyecto.

Al ser un proyecto de investigación, muchas de las cuestiones que aquí subyacen no aplican.

5.1.1 Riesgos

Los riesgos asumidos al apostar a favor del estudio y de la innovación de nuevas tecnologías relacionadas con la inteligencia artificial, dado que está en auge, son muy pequeños. La demanda excesiva de estudios relacionados con esta especialidad reduce el riesgo económico de apostar a favor de estas. Por otra parte, los imprevistos originados en estos proyectos son elevados, lo que supone un tiempo extra en el desarrollo de este.

Dicho esto, se introduce un porcentaje de error en el presupuesto inicial del 18% por imprevistos, creando un intervalo de precios acordado en la oferta.

5.1.2 Responsabilidades profesionales y éticas

Como proceso de investigación que no usa ningún tipo de dato protegido por la *Ley Orgánica de Protección de Datos de Carácter Personal del 13 de diciembre de 1999*, las responsabilidades profesionales recaen en los siguientes puntos:

1) El uso adecuado de los programas *software* posteriormente nombrados, de acuerdo con los términos y condiciones aceptados durante su instalación.
2) La honestidad y la responsabilidad que se debe poseer al realizar un proyecto de investigación. Siendo lo más objetivo y preciso que mis propias limitaciones acerca del conocimiento relacionado con el propio proyecto marcan. Para ello, se ha realizado un estudio previo del estado del arte para conocer, lo más

completamente posible, el contexto que rodea el tema de la investigación. Las cuestiones éticas acerca de la publicación de un estudio realizado desde la objetividad radican en la presencia explícita del razonamiento llevado a cabo, así como de las conclusiones extraídas en función de estas.

5.1.3 Riesgos laborales

No aplica ningún riesgo laboral en el proyecto, más allá del coste que sufraga la empresa contratante relacionado con los seguros de los trabajadores implicados en el mismo.

5.1.4 Privacidad y seguridad

No aplica ningún aspecto de privacidad ni de seguridad en el proyecto, dado que no se hace uso de ningún dato personal o relevante que aluda a cuestiones de privacidad o seguridad de información, más allá de las condiciones de seguridad que proporcionan los programas utilizados per se.

5.1.5 Rangos salariales

Los rangos salariales base del proyecto (*tabla 175*) se encuentran sujetos a dos convenios: *el Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos* (Social, Resolución de 30 de diciembre de 2016, de la Dirección General de Empleo, por la que se registra y publica el Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos., 2017), y *el Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública* (Social, Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública., 2018); ambos pertenecientes al Boletín Oficial del Estado (BOE).

El Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos suscrito con fecha 7 de noviembre de 2016 establece los rangos salariales base de los ingenieros y técnicos de las empresas en España. Este tuvo validez hasta el día 31 de diciembre del año 2017.

El Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública, suscrito con fecha 20 de noviembre de 2017 establece los rangos salariales base de los trabajadores de las empresas en España relacionados con la

consultoría, los estudios de mercado y la opinión pública. Este tiene validez desde el 1 de enero de 2010 hasta el 31 de diciembre de 2019. Este recoge las cantidades salariales del anterior convenio y posee distintas cantidades dependiendo del año.

Con motivo de establecer un salario que no incumpla ambos convenios se ha establecido la próxima tabla de salarios enfocado al proyecto en cuestión. Se ha incrementado el salario de cada oficio aproximadamente un 50% sobre el valor más alto del convenio, al tratarse de un proyecto de investigación relacionado con la inteligencia artificial, tema en auge y muy cotizado en el mercado. Este coste comprende el salario bruto, en el que se incluyen los gastos por seguridad social y la retención de IRPF aplicable. Además, se le aplicará un salario estimado al tutor debido a su rol como jefe de proyecto, y no como catedrático, pues el apartado de planificación 6.1 Planificación simula los costes de una empresa lo más cercano posible a la *realidad*.

En la siguiente tabla se muestran los siguientes aspectos:

- **Cargo**: indica la función laboral en el proyecto.

- **Máximo del salario por hora del convenio en la planificación final**: establece el valor máximo de salario por hora de los dos convenios anteriormente mencionados y de las fechas marcadas. Estos convenios protegen al trabajador ante posibles situaciones de abuso y baja renta económica. Aquel con mayor salario es el establecido a partir del 1 de abril de 2018. El cómputo se hará dividiendo el salario anual por 1800 horas, que son las que se marcan en el convenio.

- **Máximo del salario por hora del convenio en la planificación inicial**: establece el valor máximo de salario por hora del primer convenio, ya que el segundo se publicó posteriormente. El cómputo se hará dividiendo el salario anual por 1800 horas, que son las que se marcan en el convenio.

- **Salario por hora del proyecto en la planificación final**: es el resultado de aplicarle al máximo del salario por hora del convenio un incremento del 50% para sufragar los costes de seguridad social y retención del IRPF, y para no mantenerse en el mínimo valor de los decretos.

- **Salario por hora del proyecto en la planificación inicial**: es el resultado de aplicarle al salario por hora del convenio seleccionado en la planificación inicial un incremento del 50% para sufragar los costes de seguridad social y retención del IRPF, y para no mantenerse en el mínimo valor de los decretos.

Cargo	Máximo del salario por hora del convenio (€/h)		Salario por hora del proyecto (€/h)	
	Plan inicial	Plan final	Plan inicial	Plan final
Jefe de proyecto	14.30	14.59	21.45	21.88
Analista	11.15	11.38	16.72	17.07
Diseñador	11.15	11.38	16.72	17.07
Gestor de calidad y pruebas	11.15	11.38	16.72	17.07
Documentación	11.15	11.38	16.72	17.07
Programador	11.15	11.38	16.72	17.07

Tabla 175. Rangos salariales

5.2 Estándares técnicos

5.2.1 Producto final

Tratándose de un proyecto de investigación, el producto final resulta ser información contrastada acerca de las conclusiones extraídas del propio proyecto. Esa es la razón por la cual no se puede establecer ninguna aportación acerca de los estándares técnicos del producto. Sin embargo, en el punto 5.3 Propiedad intelectual se habla acerca de la patentabilidad de la idea.

5.2.2 Metodología llevada a cabo

La metodología del proyecto no está sujeta a ninguna métrica ni convenio **concretos** impuestos por el estado, como lo es métrica 3. El trabajo fin de grado, ajustado en un documento estructurado con toda la información necesaria y fundamental, junto a las matrices de corrección de este (UC3M, Informe del Tutor del Trabajo Fin de Grado, 2018)

(UC3M, Matriz de Evaluación de Trabajo Fin de Grado, 2018) y a las pautas establecidas en el punto 3.1 Metodología de estudio, establecen una metodología en cascada por cada *script* que implementar (y realizando previamente un estudio inicial de los dominios y los algoritmos a utilizar), incluyendo las etapas de análisis, diseño, implementación y pruebas, pero no la de mantenimiento. Además, cabe recalcar las recomendaciones acerca de cómo estructurar un trabajo fin de grado relacionado con la informática (UPV). Todos estos documentos se encuentran en las referencias.

5.2.3 Licencias, términos y condiciones

A continuación, se enumera la lista de dispositivos, programas y lenguajes utilizados en el proyecto. Algunos de ellos tienen términos, condiciones y licencias que es necesario comentar, junto a una breve descripción.

5.2.3.1. Herramientas *hardware*

Las herramientas hardware que se han utilizado:

- Un ordenador portátil *Asus* modelo *X550CA* (64 bits), adquirido en el mes de julio del año 2014. Su garantía finalizó a los dos años después de adquirirlo. Todos los términos acerca de la seguridad, privacidad y de licencia residen en los programas software que están instalados en el propio dispositivo.

- Un ratón de ordenador marca *Gembird* adquirido el mes de octubre de 2017, cuya garantía duró 6 meses, finalizando en el mes de abril. Ninguna cuestión de licencia aplicable al marco regulador podría relacionarse con este producto.

- Mando para presentaciones marca Doosl adquirido el mes de Septiembre de 2018, sin garantía. Ninguna cuestión de licencia aplicable al marco regulador podría relacionarse con este producto.

5.2.3.2. Herramientas *software*

Las herramientas software utilizadas son las siguientes:

- Sistema operativo *Windows 10 Home*. El ordenador portátil *Asus* veía equipado con el Sistema operativo *Windows 8.0*, pero las siguientes actualizaciones permitieron la

licencia gratuita de *Windows 10* Home para aquellos que tuviesen *Windows 8.0* o *Windows 8.1*.

Es por eso por lo que los términos de licencia para el uso de este S.O vinieron junto con el coste del propio ordenador portátil. Por otra parte, el uso de *Windows 10 Home* en este proyecto no rompe los términos y condiciones del programa.

- *Microsoft Office 365 ProPlus*. La licencia se obtuvo mediante el conjunto de programas software que la UC3M ofrece a sus estudiantes. En concreto se ha utilizado Word y Excel.

Entre los 3 tipos de licencia que ofrece *Microsoft Office* (*licencia OEM*, *licencia Retail* y *licencia por volumen*), la *Universidad Carlos III de Madrid* probablemente haya sufragado los costes de la licencia por volumen, pues está orientada a la adquisición de múltiples licencias. El coste de este programa corre a cuenta, por tanto, de la *UC3M*. Por otra parte, el uso de *Microsoft Office Home* en este proyecto no rompe los términos y condiciones del programa.

- *Adobe Acrobat Reader DC, versión 2018.011.20058*. Programa que venía de serie con *Windows*, al cual se le han ido agregando actualizaciones.

El posible coste de la licencia queda sufragado por el importe económico de la adquisición del portátil *Asus*. Por otra parte, el uso de *Adobe Acrobat Reader* en este proyecto no rompe los términos y condiciones del programa.

- *Notepad++ versión v6.8.6*. Programa *software* instalado en 2015 para la edición de ficheros de texto. La licencia de este programa es totalmente gratuita. Además, el uso de Notepad++ en este proyecto no rompe los términos y condiciones del programa.

- *Google Chrome versión 68.0.3440.106*. Programa que venía de serie con *Windows*, al cual se le han ido agregando actualizaciones. La licencia de este buscador es totalmente gratuita. Además, el uso de *Google Chrome* en este proyecto no rompe los términos y condiciones del programa. Todas las herramientas de *Google* y todos los archivos descargados y/o usados desde este navegador, como los TFGs de referencia (Maganto, 2017) (UPV), matrices de trazabilidad de corrección (UC3M, Informe del Tutor del Trabajo Fin de Grado, 2018) (UC3M, Matriz de Evaluación de Trabajo Fin de Grado, 2018), *Gmail*, *draw.io*, *Smartsheet*, etc., son de acceso público.

- Sistema operativo *Linux – Ubuntu versión 18.04*. Sistema operativo descargado de la página oficial a primeros de 2018, e instalado en el computador realizando previamente una partición del disco duro. Su licencia es gratuita, y su implicación en el proyecto no rompe sus términos y condiciones.

- Planificador *Fast Downward versión 49*. Es un planificador de licencia gratuita que puede ser instalado en *Linux*, *Mac OS X* y en *Windows*, aunque se recomienda su uso en

Linux. Este fue instalado tras instalar *Ubuntu*, es decir, a primeros de 2018. El código fuente del programa se puede cambiar y recompilar, lo que favorece los proyectos de investigación. Dicho esto, su uso y desempeño en este proyecto no rompe los términos y condiciones del programa.

- Dominios del *track clásico* (determinista) de IPC de los años 2014 y 2018 obtenidos a través de la página oficial. Constan de una serie de ficheros de texto en formato PDDL. Se puede acceder al repositorio que contiene estos ficheros de forma completamente lícita, dado que su uso es libre, con el fin de fomentar las investigaciones en planificación automática.

- Editor de texto *Gedit versión 2.30.1*. Programa que venía de serie con *Ubuntu*, al cual se le han ido agregando actualizaciones. El posible coste de la licencia queda sufragado por la adquisición del sistema operativo *Linux – Ubuntu*. De todos modos, su licencia es gratuita per se, y su implicación en el proyecto no rompe los términos y condiciones del programa.

- *Mozilla Firefox versión 60.0.2*. Programa que venía de serie con *Ubuntu*, al cual se le han ido agregando actualizaciones. El posible coste de la licencia queda sufragado por la adquisición del sistema operativo *Linux – Ubuntu*. De todos modos, su licencia es gratuita per se, y su implicación en el proyecto no rompe los términos y condiciones del programa. Todos los archivos descargados desde este navegador, como los TFGs de referencia (Maganto, 2017) (UPV), matrices de trazabilidad (UC3M, Informe del Tutor del Trabajo Fin de Grado, 2018) (UC3M, Matriz de Evaluación de Trabajo Fin de Grado, 2018), etc., son de acceso público.

5.2.3.3. Lenguajes

Los lenguajes de programación y planificación son los siguientes:

- *Python*. Lenguaje de programación incorporado en *Ubuntu*. Se puede hacer libre uso de él.

- C++. Lenguaje de programación incorporado en *Ubuntu*. Al igual que *Python*, se puede usar este lenguaje de forma gratuita.

- PDDL. Lenguaje de planificación que interpreta el planificador *Fast Downward*. Todos los posibles términos, condiciones y licencias de este lenguaje los recoge el propio planificador.

5.3 Propiedad intelectual

Todos los proyectos fin de grado están protegidos por la *Ley de Propiedad Intelectual del 12 de abril de 1996*, y por la *Ley de Patentes del 20 de marzo de 1986*. Estas leyes señalan que la titularidad de los derechos de propiedad intelectual de los trabajos fin de grado corresponde a los autores del proyecto, es decir, a los estudiantes que lo hayan realizado. Igualmente, podría compartirse dicha titularidad con los tutores y entidades que hayan participado en el mismo. Por otra parte, a pesar de que el autor autorice a la universidad su consulta y exposición pública, esto no menoscaba los derechos de autor (wikis.fdi.ucm, 2016).

En este caso particular, todas las ideas y conclusiones aquí recogidas están protegidas por dicha *Ley de Propiedad Intelectual*. Dicho esto, es lícito el hecho de usar la información aquí presente, siempre y cuando deje constancia, en sus referencias, de la fuente original.

SECCIÓN 6. ENTORNO SOCIO-ECONÓMICO

En esta sección se establecerá la planificación inicial y final del proyecto, junto a sus desviaciones, el presupuesto inicial que se le dará al cliente, el coste total y final del proyecto y el entorno socio-económico de este.

6.1 Planificación

En los siguientes 3 puntos se indicará la planificación inicial (*tabla 176 e ilustraciones 7 – 9*) y final (*tablas 177 y 178 e ilustraciones 10 – 13*) del proyecto, junto a sus desviaciones.

Cabe indicar que el periodo de la planificación comprende desde la primera reunión con el tutor hasta el momento de la entrega del proyecto. Es decir, no se contabilizan las horas destinadas a las tutorías con el resto de los profesores en busca del tema del Trabajo Fin de Grado, ni el tiempo destinado a la presentación del proyecto.

6.1.1 Planificación inicial

- **Horas totales estimadas**:

Actividad	Fecha inicio	Fecha final	Horas totales estimadas
Reuniones con el tutor	-	-	15 horas
Estudio inicial del tema del TFG	23/10/17	2/11/17	3 horas
Elección del TFG, trámite y planteamiento de objetivos	3/11/17	16/11/17	8 horas
Planificación y presupuesto iniciales	16/11/17	17/11/17	5 horas
Estudio personal de la materia (estado del arte): aprendizaje de C++, PDBs, uso de *Fast Downward*, etc.	18/11/17	25/12/17	100 horas
Metodología, diseño y requisitos	26/12/17	8/1/18	50 horas

Heurísticas dependientes del dominio	9/1/18	24/2/18	5 dominios*50 horas por dominio = 250 horas
Estudio independiente del dominio	25/2/18	5/3/18	75 horas
Extracción de conclusiones	6/3/18	6/3/18	10 horas
Documentación final	7/3/18	12/6/18	200 páginas estimadas * una página por 1,5 horas = 300 horas
HORAS TOTALES	-	-	**816 horas**

Tabla 176. Planificación inicial - horas totales

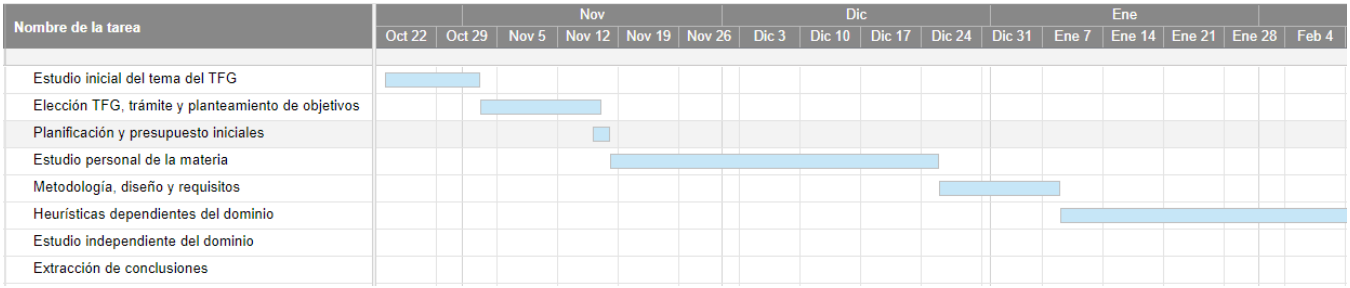

Ilustración 8. Gantt plan inicial 1

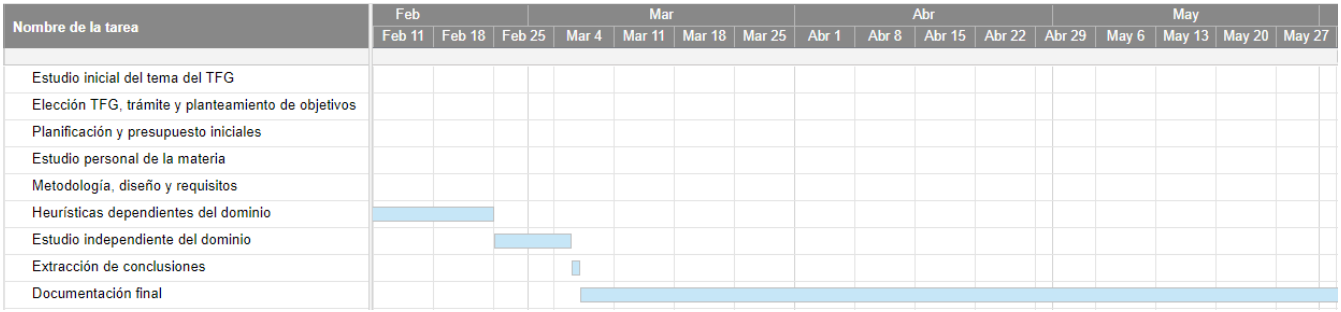

Ilustración 7. Gantt plan inicial 2

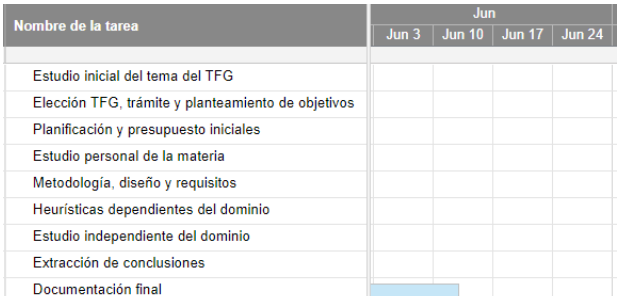

Ilustración 9. Gantt plan inicial 3

- **Deviaciones en el horario permitidas en la oferta**: siempre y cuando el proyecto finalice, como muy tarde, en la tercera sesión (25/9/2018).

- **Desviaciones en el número total de horas permitido en la oferta**: no hay máximo de horas específico. Este lo determina el presupuesto. Por otra parte, se estima que supera las 300 horas mínimas destinadas a la realización del TFG.

6.1.2 Planificación final

- Las horas referentes a las **reuniones** con el tutor / jefe de proyecto son las siguientes:

Número de reunión con el tutor / jefe de proyecto	Fecha de la reunión	Duración de la reunión
Primera	23/10/2017	50 minutos
Segunda	16/11/2017	55 minutos
Tercera	21/11/2017	45 minutos
Cuarta	5/2/18	45 minutos
Quinta	19/3/18	1 hora
Sexta	26/6/18	1 hora
Séptima	4/7/18	45 minutos

Octava	11/7/18	45 minutos
Novena	18/7/18	1 hora y 10 minutos
Décima	23/7/18	1 hora
Undécima	27/7/18	50 minutos
Duodécima	6/9/18	50 minutos
HORAS TOTALES	-	**10 horas y 35 minutos**

Tabla 177. Planificación final - reuniones

- **Horas totales reales**:

Actividad general	Actividad desglosada	Fecha inicio	Fecha final	Horas totales estimadas
Estudio inicial del tema del TFG	-	23/10/17	2/11/17	3 horas
Elección del TFG, trámite y planteamiento de objetivos	-	3/11/17	16/11/17	8 horas
Planificación y presupuesto iniciales	-	16/11/17	17/11/17	8 horas
-	**Planificación inicial**	16/11/17	-	4 horas
-	**Presupuesto inicial**	-	17/11/17	4 horas
Estudio y preparación previa de la materia	-	18/11/17	19/3/18	146 horas
-	**Aprendizaje de C++**	18/11/17	-	43 horas
-	**Repaso de planificación automática y PDDL**	-	-	5 horas

-	**Instalación y configuración de *Ubuntu***	-	-	7 horas
-	**Instalación de Fast Downward**	-	-	4 horas
-	**Comprensión de Fast Downward y PDBs**	-	-	41 horas
-	**Búsqueda, descarga y selección de los dominios de IPC**	-	-	14 horas
-	**Comprensión de los dominios de IPC**	-	-	10 horas
-	**Búsqueda y comprensión de trabajos relacionados**	-	-	13 horas
-	**Estudio del contenido exigido por la UC3M en un TFG**	-	-	2 horas
-	**Estudio del marco regulador**	-	19/3/18	7 horas
Metodología, diseño y requisitos	-	19/3/18	28/6/18	24 horas
-	**Establecimiento de la metodología de estudio**	19/3/18	-	13 horas
-	**Establecimiento de las plantillas de requisitos y casos de uso**	-	-	3 horas
-	**Establecimiento inicial de requisitos y casos de uso**	-	28/6/18	8 horas

Dominio *Snake* (experimentación y análisis de resultados)	-	29/6/18	30/7/18	53 horas
Dominio *Termes* (experimentación y análisis de resultados)	-	30/6/18	30/7/18	68 horas
Replanificación y refinamiento de la metodología	-	31/7/18	31/7/18	6 horas
-	**Replanificación**	31/7/18	-	4 horas
-	**Refinamiento de la metodología**	-	31/7/18	2 horas
Dominio *Hiking* (experimentación y análisis de resultados)	-	31/7/18	3/8/18	31 horas
Dominio *Tetris* (experimentación y análisis de resultados)	-	3/8/18	6/8/18	36 horas
Dominio *Spider* (experimentación y análisis de resultados)	-	6/8/18	10/8/18	45 horas
Estudio independiente del dominio (experimentación y análisis de resultados)	-	10/8/18	16/8/18	61 horas
Extracción de conclusiones	-	17/8/18	17/8/18	9 horas
Documentación final	-	18/8/18	10/9/18	306 horas
-	**Apartados, estilo y formato**	18/8/18	-	5 horas

-	**Portada, frase inicial, agradecimientos, resumen**	-	-	4 horas
-	**Introducción**	-	-	10 horas
-	**Estado del arte**	-	-	56 horas
-	**Marco regulador**	-	-	10 horas
-	**Entorno socio-económico (incluyendo la planificación y el presupuesto finales)**	-	-	19 horas
-	**Diseño de la solución (añadiendo los requisitos restantes y la definición de las funciones)**	-	-	61 horas
-	**Investigación**	-	-	96 horas
-	**Conclusiones**	-	-	3 horas
-	**Trabajos futuros**	-	-	4 horas
-	**Anexo (abstract)**	-	-	8 horas
-	**Formato de las referencias**	-	-	1 hora
-	**Repasar todo, añadirle formato, corregir faltas de ortografía… (versión 1)**	-	10/9/18	29 horas
Establecimiento de la versión 2 de la memoria tras la corrección de erratas	-	12/9/18	25/9/18	41 horas
HORAS TOTALES	**-**	**-**	**-**	**845 horas**

Tabla 178. Planificación final - horas totales

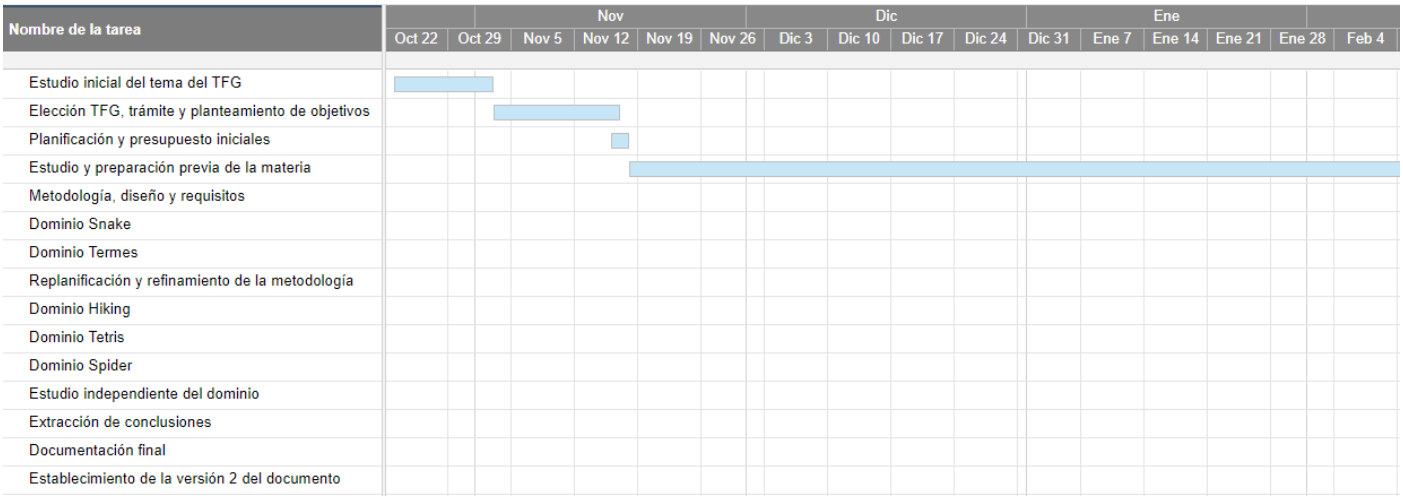

Ilustración 12. Gantt plan final 1

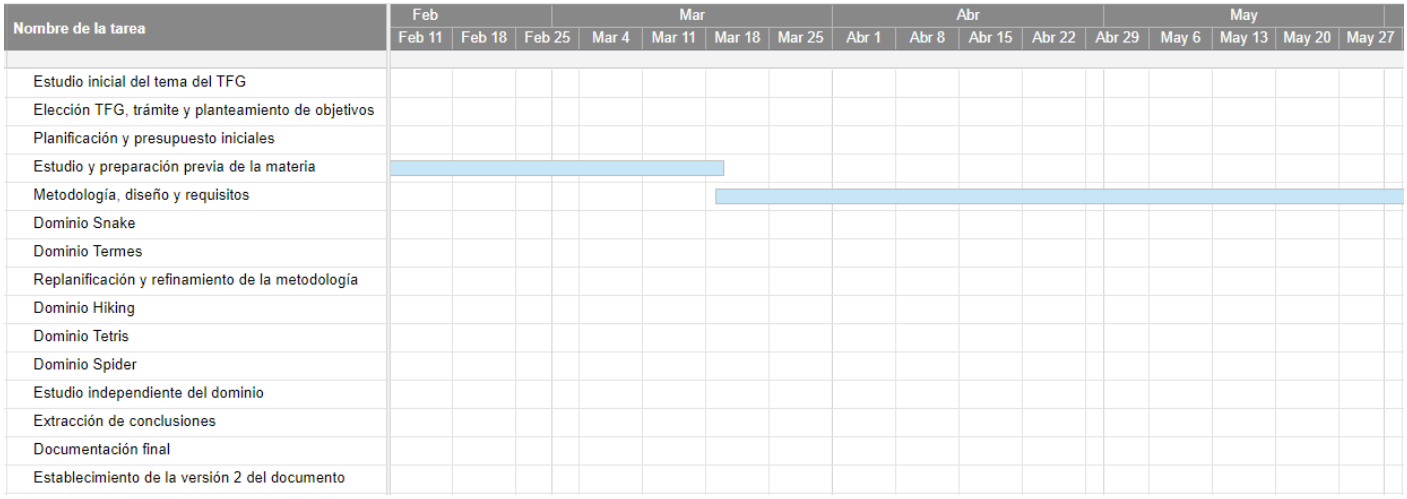

Ilustración 11. Gantt plan final 2

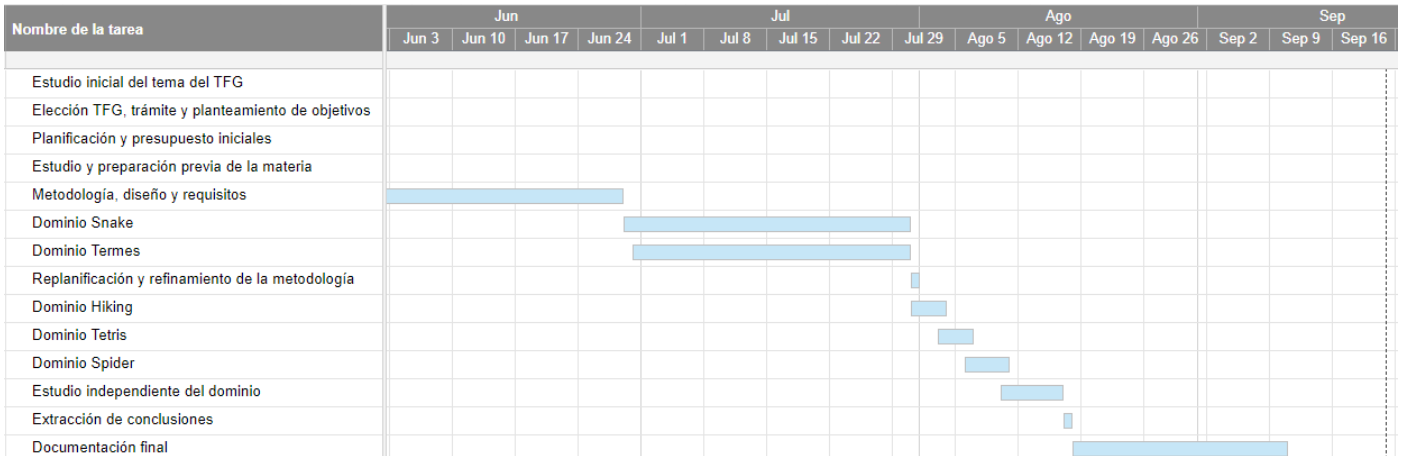

Ilustración 10. Gantt plan final 3

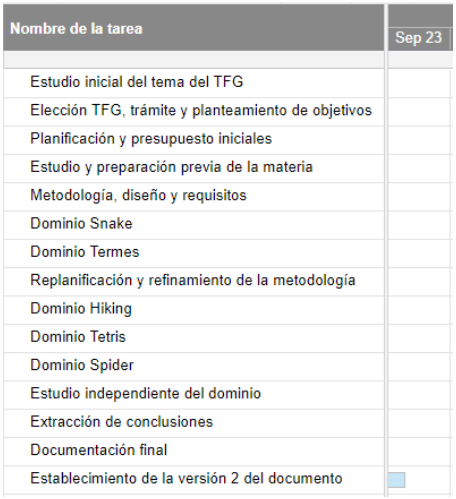

Ilustración 13. Gantt plan final 4

Contabilizando también las reuniones, el tiempo dedicado al proyecto ha sido de 855 horas y 35 minutos, redondeado a **856 horas**. El TFG equivale a 300 horas de trabajo contando la presentación, cumpliendo con el esfuerzo mínimo que hay que llevar a cabo.

Las horas dedicadas a la presentación no se van a contabilizar ni en el presupuesto ni en la planificación ya que son desconocidas, pero el coste del mando para las transparencias, el viaje hasta allí para entregar la autorización de publicación del TFG y para presentarlo y la vestimenta de la presentación sí. En resumidas cuentas, todo suma a excepción del coste de las horas dedicadas a preparar la presentación, el tiempo de redacción de e-mails y el tiempo de ejecución del portátil mientras descansaba.

Las ideas que surgían acerca de trabajos futuros, la recolección de referencias el cómputo de horas de la planificación final y el presupuesto final se realizó durante el transcurso del proyecto. Por otra parte, el resto de material del anexo corresponde al proceso de investigación.

6.1.3 Desviaciones

Por circunstancias personales, la planificación final del proyecto dista mucho de la planificación final. Sin embargo, esto no supone un coste adicional significativo, pues las horas mínimas destinadas al proyecto siguen siendo 300 y el tiempo dedicado y los costes que acarrean son similares (aplicando un margen de error). Solo cambia la planificación, que sigue estando dentro de los límites de la entrega del proyecto, que pasa a ser de la segunda a la tercera sesión.

En resumidas cuentas, la planificación final entra dentro de los límites de la entrega, que es hasta la tercera sesión, acordado en la oferta. Respecto al coste, se estableció un intervalo de variación del coste del proyecto, que también se acordaría en la oferta. Las desviaciones, por tanto, no han ocasionado pérdidas en el beneficio de la empresa ni ha roto el acuerdo del cliente.

6.2 Presupuesto

A continuación, se indicará el coste inicial y final del proyecto. Este se desglosa en costes salariales, dispositivos *hardware* y complementos, herramientas *software*, material fungible, viajes y dietas, costes indirectos, impuestos, beneficios y riesgos.

6.2.1 Costes salariales

- Antes de establecer los costes, es necesario aclarar algunos puntos:

- ⇨ Las **reuniones** con el tutor suponen un rol de analista, aunque en muchas ocasiones dichas reuniones implicaron una formación acerca de la materia en lugar de una mera reunión con uno de los jefes del proyecto. A efectos del presupuesto, así se va a contabilizar.
- ⇨ El papel de **jefe de proyecto** se va a aplicar en las siguientes actividades:
  - Las actividades realizadas por el tutor Daniel Borrajo Millán.
  - Estudio inicial del tema del TFG.
  - Elección del TFG, trámite y planteamiento de objetivos.
  - Planificación y presupuestos iniciales
  - Estudio y preparación previa de la materia – Estudio del contenido exigido por la UC3M en un TFG.
  - Estudio y preparación previa de la materia – Estudio del marco regulador.
  - Documentación final – Apartados, estilo y formato.
  - Documentación final – Portada, frase inicial, agradecimientos, resumen.
  - Documentación final – Marco regulador.
  - Documentación final – Entorno socio-ecnonómico.
  - Documentación final – Anexo (abstract)
  - Documentación final – Formato de las referencias.
  - Documentación final – Repasar todo, añadirle formato, corregir faltas de ortografía...
  - Establecimiento de la versión 2 del documento.
- ⇨ Respecto al cómputo de horas en la **planificación inicial** dedicadas por el jefe de proyecto al estudio personal de la materia y a la documentación final este será del 15% del peso total.

⇨ Las **horas del tutor** / jefe de proyecto se van a extraer de las horas destinadas a las reuniones, incrementando un 75% de horas destinadas a un trabajo exclusivo del tutor relacionado con su implicación individual en el proyecto. Con el fin de amortiguar el coste en la lectura de artículos, lectura del proyecto final, estudio de posibles objetivos, etc.

– Dicho esto, el coste salarial inicial es:

Rol	Persona	Salario por hora (€/h)	Número de horas totales (h)	Salario total (€)
Jefe de proyecto	Daniel Borrajo Millán	21.45	26.25	563.07
	José González Barroso	21.45	76	1630.20
Analista, diseñador gestor de calidad y pruebas, documentación y programador	José González Barroso	16.72	740	12372.80
Coste total de personal	-	-	-	**14566.07**

Tabla 179. Coste salarial inicial

- Por otra parte, el coste salarial final es:

Rol	Persona	Salario por hora (€/h)	Número de horas totales (h)	Salario total (€)
Jefe de proyecto	Daniel Borrajo Millán	21.88	18.52	405.22
	José González Barroso	21.88	145	3172.60
Analista, diseñador gestor de calidad y pruebas, documentación y programador	José González Barroso	17.07	711	12136.77
Coste total de personal	-	-	-	**15714.59**

Tabla 180. Coste salarial final

6.2.2 Dispositivos hardware y complementos

Dispositivos hardware o complemento	Unidades	Coste unitario (€)	Desde su adquisición, porcentaje de implicación en el proyecto	Coste (€)
***Portátil Asus*, modelo *X550CA* de 64 bits**	1	649.90	13% (25% por año)	84.49
Ratón marca *Gembird*	1	19.99	60%	12.00
Alfombrilla para el ratón marca *ALLSOP*	1	2.99	60%	1.80

Mando para presentaciones marca *Doosl*	1	13.90	100%	13.90
Vestimenta para la presentación	1	175	30%	52.50
Coste total		-	-	**164.69**

Tabla 181. Coste de dispositivos hardware y complementos

- Puesto que se desconocen los recursos utilizados por el tutor, se establece un 10% adicional del coste total por la utilización de sus dispositivos y complementos, es decir, **16.47 €**.

- El coste en el presupuesto final e inicial coinciden.

6.2.3 Herramientas *software*

Dispositivos *software*	Unidades	Coste unitario (€)	Coste (€)
Sistema operativo *Windows 10 Home*	1	0.00	0.00
Microsoft Office 365 ProPlus	1	0.00	0.00
Adobe Acrobat Reader DC, versión 2018.011.20058	1	0.00	0.00
Notepad++ versión v6.8.6	1	0.00	0.00
Google Chrome versión 68.0.3440.106	1	0.00	0.00
Sistema operativo *Linux – Ubuntu versión 18.04*	1	0.00	0.00
Planificador *Fast Downward versión 49*	1	0.00	0.00

Editor de texto *Gedit versión 2.30.1*	1	0.00	0.00
Mozilla Firefox versión 60.0.2	1	0.00	0.00
Coste total	-	-	**0.00**

Tabla 182. Coste de herramientas software

- El coste de *Microsoft Office 365 ProPlus* corre a cuenta de la Universidad, suponiendo que no tiene implicación directa en el aporte de los alumnos en la realización de la matrícula.

- El coste en el presupuesto final e inicial coinciden.

6.2.4 Material fungible

Material fungible	Unidades	Coste unitario (€)	Coste (€)
Paquete de 100 folios	1	2.00	2.00
Bolígrafos	3	0.42	1.26
Corrector	1	1.50	1.50
Lápices HB	2	0.22	0.44
Borrador	1	0.11	0.11
Sacapuntas	1	1.70	1.70
Paquete de 20 fundas de plástico transparentes para folios	1	2.50	2.50
Coste total	-	-	**9.51**

Tabla 183. Coste del material fungible

- Puesto que se desconocen los recursos utilizados por el tutor, se establece un 10% adicional del coste total por la utilización del material fungible, es decir, **0.96 €**.

- No se aplica porcentaje de amortización.

- **Coste inicial**: estimación de **15.00 €**. Por tanto, se estima un coste de **1.50 €** por el tutor.

6.2.5 Viajes y dietas

Viajes y dietas	Unidades	Coste unitario (€)	Amortización	Coste (€)
Abono transporte	8	20.00	10%	16.00
Dietas	0	-	-	0.00
Coste total	**-**	**-**		**16.00**

Tabla 184. Coste de viajes y dietas

- Puesto que se desconocen los recursos utilizados por el tutor, se establece un 25% adicional del coste total por viajes y sietas, es decir, **4.00 €**.

- **Coste inicial**: estimación de **30.00 €**. Por tanto, se estima un coste de **7.50 €** por el tutor.

6.2.6 Costes indirectos

- Los costes indirectos son una **estimación**. La diferencia entre el presupuesto inicial y final radica en los meses dedicados a la realización del proyecto.

- Aunque este coste se suela obtener mediante el 15/20% del coste del personal, dada las condiciones del proyecto, conllevaría a una sobreestimación del coste. Por tanto, un desglose sería más adecuado.

- El coste indirecto inicial es:

Viajes y dietas	Unidades	Coste unitario (€)	Amortización	Coste (€)
Abono transporte	8	20.00	10%	16.00
Dietas	0	-	-	0.00
Coste total	**-**	**-**		**16.00**

Tabla 185. Costes indirectos iniciales

- El coste indirecto final es:

Costes indirectos	Meses	Coste unitario (€)	Amortización por mes	Coste (€)
Electricidad	12	42	10%	50.40
Agua	12	20	10%	24.00
Internet	12	15	25%	45.00
Coste total	-	**-**	-	**119.40**

Tabla 186. Costes indirectos finales

- Puesto que se desconocen los recursos utilizados por el tutor, se establece un 5% adicional del coste total por viajes y sietas, es decir, **4.48 €** en la **planificación inicial** y **5.97 €** en la **final**.

6.2.7 Presupuesto *inicial* del proyecto

Concepto	Coste (€)
Costes salariales	14566.07
Dispositivos hardware y complementos	164.69 + 16.47
Herramientas *software*	0.00

Material fungible	15.00 + 1.50
Viajes y dietas	30.00 + 7.50
Costes indirectos	89.55 + 4.48
Coste total	**14895.36**

Tabla 187. Coste total inicial

Concepto	**Cantidad monetaria (€)**
Gasto total del proyecto	14895.36
Gasto total de la presentación	GRATIS
Beneficio (20%)	2979.08
TOTAL SIN I.V.A	**17874.44**
I.V.A (21%)	3753.64
TOTAL CON I.V.A	**21628.08**

Tabla 188. Precio total inicial

El coste estimado del proyecto es de **21628.08€**. Se establece un rango de error del **18%**.

En la oferta se propone un proyecto que oscila entre los **17735.03 €** y los **25521.14 €**. Respecto a la planificación, se estima que el proyecto, exceptuando la presentación, finalizará el día **12/6/18**. Sin embargo, se da la posibilidad de que este se postponga hasta antes de la finalización de la entrega en la tercera sesión, es decir, el día **25/9/18**. Se establecería una firma para legitimar el acuerdo con el cliente.

6.2.8 Coste total y final del proyecto

Concepto	Coste (€)
Costes salariales	15714.59
Dispositivos hardware y complementos	164.69 + 16.47
Herramientas *software*	0.00
Material fungible	9.51 + 0.96
Viajes y dietas	16.00 + 4.00
Costes indirectos	119.40 + 5.97
Coste total	**16051.59**

Tabla 189. Coste total final

Concepto	Cantidad monetaria (€)
Gasto total del proyecto	16051.59
Gasto total de la presentación	GRATIS
Beneficio (20%)	3210.32
TOTAL SIN I.V.A	**19261.91**
I.V.A (21%)	4045.01
TOTAL CON I.V.A	**23306.92**

Tabla 190. Precio total final

El coste final del proyecto es de **23306.92€** y finaliza el día **25/9/18**, cumpliendo el acuerdo con el cliente establecido en la oferta.

6.3 Impacto socio-económico

En una era donde la tecnología impregna cada ámbito de la sociedad actual, es innegable el hecho de que se requiere, a su vez, de avances tecnológicos. Esto alude también a la necesidad de realizar investigaciones para encontrar nuevos modelos matemáticos, algoritmos; implementar nuevos lenguajes que satisfagan otras necesidades, etc.

Además, dentro de estas nuevas tecnologías, la inteligencia artificial es una de las más comentadas y requeridas. Nadie se extraña al escuchar que la informática es vital para el futuro que nos depara, pero nada más lejos de la realidad, también es vital para el presente. Es por eso por lo que los proyectos de investigación, avances o aplicaciones relacionados con esta disciplina son tan demandados.

Ahora bien, las entidades que podrían demandar proyectos de este estilo son:

1) Empresas privadas que requieran de información de este tipo para sacar al mercado productos más eficientes y más eficaces.
2) Organismos públicos, invirtiendo en I+D+I para la prosperidad económica y social de su localidad, región o nación.
3) Empresas dedicadas a la publicación de investigaciones tecnológicas.
4) Personas físicas que requieran de esta información para la redacción de artículos, renombre, tesis, trabajos, etc.

El proyecto actual se relaciona con la investigación en la planificación automática, que es una disciplina de la inteligencia artificial. Saber esto incrementa el valor del proyecto, pues, como se ha comentado en los dos anteriores párrafos, la inteligencia artificial está en auge.

Dicho esto, es necesario destacar la importancia de realizar mejoras, concretamente, en planificación automática. Todas estas aplicaciones podrían tener un impacto en mayor o menor medida sobre el medioambiente o la ética, pero dependen exclusivamente de la aplicación práctica en sí, no de la planificación automática. A continuación, se enumeran **algunas** aplicaciones:

1) Vehículos autónomos para logística y envíos.
2) Gestión de almacenes.
3) Automatización de cadenas de producción.
4) Exploración submarina y espacial.
5) Planificación de drones.
6) Tomas de decisiones con el mínimo coste en empresas.
7) Establecimiento de rutinas de actuación en caso de emergencia.

Como se ha ido observando, el proyecto tiene valor de cara a las empresas. A nivel social, cualquier avance tecnológico influye directamente en la calidad de vida de las personas.

Y, por último, a nivel económico también se espera una gran influencia, pues toda empresa busca obtener beneficio, y las personas desean obtener comodidad con el menor costo posible.

El avance de la tecnología, en general, satisface tanto a las empresas como a la sociedad en general. Dicho esto, todas las aplicaciones que podría tener la planificación automática mejorarán el nivel económico en cada una de las empresas que hagan un uso apropiado de esta disciplina, y el nivel de comodidad y seguridad en la sociedad; aún más cuando se realizan investigaciones para potenciar estas herramientas y disciplinas.

SECCIÓN 7. CONCLUSIONES GENERALES

Durante el proceso de investigación Sección 4 Investigación se han redactado numerosas conclusiones observando los resultados de los experimentos. En esta sección se mostrarán aquellas conclusiones **generales** que se pueden extraer del proyecto.

La **primera** conclusión es que, generalmente, los mejores algoritmos con los que se ha experimentado son los implementados relacionados con una heurística dependiente del dominio (mediante *Canonical*). Ya sea por el tiempo ahorrado en la generación de patrones, en el peor de los casos, o por la calidad de la heurística obtenida. Se puede afirmar la calidad de las heurísticas que proporciona dicho algoritmo debido a que los dominios que se han escogido son muy distintos entre sí. A estos les siguen cualquiera de los iPDB (el de *Fast Downward* o el modificado para la heurística independiente del dominio). Sin embargo, su efectividad depende de las características del problema y del dominio en sí. De forma empírica se ha visto que trae mejores resultados (de nuevo en términos generales) la interpretación de la heurística de *Canonical* que la de *Zero-One*.

Dicho esto, una buena metodología de trabajo podría consistir en, sobre un problema que se quiera resolver, ejecutar algún algoritmo *combo* (*Canonical* o *Zero-One*), los que funcionan mejor con poblemas extremadamente sencillos. Si este no resuelve el problema, ejecutar *iPDB* y/o *iPDB* modificado. Si aún sigue sin resolverse, realizar un estudio de las variables relevantes del dominio, implementar un algoritmo dependiente del dominio y ejecutarlo mediante *Canonical PDB*. Y, en última instancia, por si *Canonical PDB* mostrase resultados bastante peores que *iPDB*, intentar ejecutar el algoritmo dependiente del dominio mediante *Zero-One*. Sin embargo, aún quedan algoritmos relacionados con PDB que analizar, como los algoritmos genéticos en *Canonical* y en *Zero-One*. Esto puede ser base de futuras investigaciones.

Una explicación del éxito de estos algoritmos implementados es que, además del tiempo ahorrado en la construcción de patrones, la investigación realizada se centra en identificar características similares en la formación de PDBs. De esta forma, al generalizar, se es capaz de alcanzar patrones a los que *iPDB* aspiraba pero que no terminó de alcanzar debido a su forma de trabajar y de detener el proceso de obtención de PDBs. Por otra parte, debido a esta generalización, ciertas ejecuciones sobre ciertos problemas originaban peores resultados que *iPDB*, ya que este último se adapta mejor a las caracterícticas concretas del dominio y del problema en sí.

Para comparar la potencia de estos algoritmos implementados, es importante nombrar que tanto *AlgSnake* en *Canonical* como *AlgTetris* en *Canonical* resuelven más problemas que los mejores planificadores utilizados en las competiciones de IPC. *AlgSnake* el *Canonical* resuelve 16 problemas, y el mejor planificador de la competición, 14. Por otra parte, *AlgTetris* resuelve 12, y el mayor número de problemas resueltos por un planificador en la competición fue de 11.

La **segunda** conclusión da respuesta a la pregunta de si merece la pena realizar un estudio de las PDBs proporcionadas por algoritmos como *iPDB* para implementar un algoritmo. La respuesta es un sí rotundo. Aunque vale la pena recalcar que depende del problema en sí. Hay que plantear siempre si es rentable invertir tiempo en el desarrollo de un algoritmo que proporcione heurísticas dependientes del dominio en un dominio concreto, observando el número de problemas que se quiere resolver o cómo de bien funcionan los algoritmos ya implementados sobre estos dominios y problemas. La calidad del algoritmo implementado está íntimamente relacionada con la precisión de la investigación. Se ha observado que no es tan necesario generalizar, siempre y cuando se adapte al dominio y a los problemas; por lo que estaría bien, en futuros proyectos, comparar más de un algoritmo dependiente del dominio entre sí. Por otra parte, incluir otros algoritmos en el proceso de observación previa a la implementación también podría ser una gran idea.

La **tercera** conclusión se relaciona con el término de predicados *relevantes*, y es que parece ser que aquellas variables que funcionan bien en PDBs son aquellas relacionadas con los predicados que aparecen en las precondiciones de todas las acciones cuyos efectos contengan un predicado relacionado con las metas del problema a resolver. Sin embargo, esto no es generalizable y puede ser desmentido mediante los resultados de futuras investigaciones que abarquen esta temática.

La **cuarta** conclusión da respuesta a la pregunta de si merece la pena investigar acerca de heurísticas independientes del dominio. En primer lugar, a pesar de que los dos algoritmos iPDB sean similares, uno destaca por encima del otro en ciertas ocasiones. Si bien es cierto que la investigación realizada no ha dado como resultado un algoritmo potente, el hecho de que destaque por encima de *iPDB* en algunos casos es una clara indicación de que sería una buena idea dedicar más tiempo en desarrollar una heurística independiente del dominio mediante más estudios. Durante la investigación realizada, se ha hablado de posibles futuros proyectos. En la Sección 8 Trabajos futuros se redactan trabajos que siguen con esta misma línea de investigación y que podrían mejorar, incluso, los resultados de este proyecto. En resumidas cuentas, se ha demostrado que los resultados de *iPDB* se pueden mejorar desarrollando una heurística independiente del dominio.

La **quinta** y última conclusión es necesaria para encasillar el proyecto realizado. Observando el gran número de trabajos futuros propuestos y de los buenos resultados obtenidos, este proyecto podría catalogarse como base. Todas las futuras investigaciones que partan de estos resultados y propuestas deberían tener el objetivo de mejorar, profundizar y/o desmentir lo que aquí se ha abordado. En el fondo, la calidad de este proyecto radica, en gran parte, de las propuestas redactadas en la sección Sección 8 Trabajos futuros. Por otra parte, el impacto socio-económico que conlleva este estudio se comenta 6.3 Impacto socio-económico .

Cabe destacar que todas estas investigaciones se han realizado sobre dominios que funcionaban bien con PDBs. En caso contrario, no significa que las conclusiones aquí alcanzadas no sean aplicables sobre estos dominios, solo que haría falta realizar experimentos sobre estos antes de emitir un juicio objetivo.

7.1 Dificultades personales y errores cometidos

Debido a la complejidad del proyecto, ya que requería un proceso de aprendizaje previo demasiado extenso, la mayor parte de los errores y dificultades del proyecto se deben a la inexperiencia. En los primeros meses de trabajo, surgieron muchas dudas y se tomaron no muy buenas decisiones. A medida que el proyecto avanzaba, estas dudas se iban solventando y las malas decisiones corrigiendo. De forma general, estos estaban relacionados con el funcionamiento de *iPDB*, la forma de trabajar con *Fast Downward* y de la verdadera utilidad del conjunto de patrones. El tutor Daniel Borrajo Millán y el trabajo personal solventaron estas dificultades.

Estas dificultades conllevaron a romper la planificación inicial del proyecto, pues era necesario entender bien estas cuestiones antes de avanzar con la investigación en cuestión. Las asignaturas presentes durante el curso también ocasionaron esta replanificación del proyecto.

7.2 Relación del trabajo con los estudios cursados

En el tercer año de ingeniería informática en la *UC3M* es necesario escoger una de las tres menciones que ofrecen: sistemas de información, computadores o computación. Aquella orientada a la inteligencia artificial, complejidad computacional, *Big Data*… es computación, y fue la que, como alumno, escogí. El proyecto actual se relaciona, directamente, con la inteligencia artificial, pues la planificación automática es una disciplina dentro de esta. Además, todos los aspectos relacionados con la documentación, planificación y presupuesto están íntimamente relacionados con las asignaturas *Fundamentos de Gestión Empresarial*, *Principios de Desarrollo de Software*, I*ngeniería del Software* y *Dirección de Proyectos de Desarrollo de Software*.

Por otra parte, en las asignaturas *Ingeniería del Conocimiento* e *Inteligencia Artificial en la Industria del Entretenimiento* se imparten unas nociones básicas acerca de la planificación automática relacionadas con el aprendizaje del lenguaje PDDL y con la mera ejecución de los planificadores. El actual proyecto se adentra en el interior del planificador. Esto se imparte en el *Máster de Ciencia y Tecnología Informática* de la *UC3M*, cuyo coordinador es Daniel Borrajo Millán, tutor de este TFG.

A medida que mis estudios en la planificación automática avanzaban, observaba que muchas bases teóricas aprendidas en otras asignaturas se aplicaban directamente en esta disciplina. Desde algoritmos de búsqueda como *Hill climbing* y *A**, pasando por teoría de conjuntos y desarrollo de heurísticas.

A pesar de que este proyecto se encasille mejor como un trabajo de máster, la base teórica aprendida durante la carrera fue suficiente para comprender la planificación automática sin ningún tipo de problema. Sin esa base teórica que el grado me ha proporcionado, no hubiese podido afrontar las exigencias de este proyecto, al menos con la profesionalidad y madurez que un proyecto de esta embergadura exige.

7.3 Valores y conocimiento adquiridos

El conocimiento adquirido durante este trabajo es enorme. Como ya se ha comentado, el temario abordado es de máster, por lo que es necesario repasar aspectos acerca de la planificación automática ya aprendidos durante los cuatro años de formación en la universidad. Además, fue necesario aprender cuestiones acerca de cómo se implementan los planificadores, cómo funcionan, cómo implementar las heurísticas, diferentes tipos de heurística, etc. Respecto a las herramientas y lenguajes utilizados, predominan el aprendizaje de *C++* (el cual desconocía en gran parte, lo que ha supuesto invertir mucho tiempo en aprenderlo) y el funcionamiento del planificador *Fast Downward*. El dominio y soltura adquiridos tras este aprendizaje ha conllevado poder realizar un proceso de investigación de forma muy objetiva ateniéndome a los resultados obtenidos y a la metodología llevada a cabo. Hasta el punto de llegar a superar algunas marcas de los mejores planificadores de la IPC.

Respecto a los valores adquiridos durante la realización de este proyecto destacan la iniciativa y la inquietud por poder desempeñar más procesos de investigación en un futuro, así como las ganas de redactar artículos basándose en dichas investigaciones. Aun sabiendo lo duro que es llevar a cabo un proyecto así, considero que puedo aportar bastante a esta disciplina.

SECCIÓN 8. TRABAJOS FUTUROS

En esta sección se propondrán futuros trabajos de investigación de la misma línea que el actual proyecto. Durante el transcurso de este documento se han ido citando diversas ramas de investigación a raíz en este trabajo. Aquí se recopilan las más relevantes, aunque es posible la existencia de otras aquí no nombradas. Proponer posibles proyectos que continúen con este es muy importante, casi tanto como las conclusiones obtenidas.

Antes que nada, es necesario referenciar el apartado 3.2 Alternativas en la metodología de estudio, donde se mencionan potenciales trabajos que podrían surgir tras variar la metodología de trabajo. De todos aquellos que se mencionan, se recomiendan los del apartado 3.2.2 Modificaciones drásticas, ya que los otros no muestran aparentemente un gran cambio respecto al actual proyecto.

Posibles proyectos futuros podrían basarse en realizar esta misma metodología de experimentación, pero tomando otras decisiones. De esta forma, los resultados de varios proyectos con variaciones en dichas decisiones convergerían a un conocimiento más profundo de la materia, siempre y cuando se comparen y se contrasten los unos con los otros. Estas decisiones podrían ser:

- Probar otros algoritmos, como el *Merge and Shrink* o los algoritmos genéticos. Este primero suele tardar mucho tiempo al ejecutarlo. Seleccionarlo para futuras investigaciones podría ser una muy buena idea. El segundo no es aconsejable personalmente, pues suele dar error. Una mejor idea sería implementarlo desde cero.

- Probar otros dominios y problemas. Sin embargo, probar la heurística independiente del dominio obtenida en este proyecto en dominios distintos no proporcionará unos grandes resultados, a no ser que no funcione bien el uso de PDBs en esos nuevos dominios en cuestión. Ya se ha analizado la calidad de esta heurística, y se intuye que el funcionamiento es el mismo, pues se han extraído las conclusiones a partir de dominios muy distintos entre sí. Además, es muy similar a *iPDB*, y este último funciona bien en dominios donde funcionan bien los patrones.

- Tomar otras alternativas en el desarrollo de la heurística independiente del dominio parecidas a las tomadas en el proyecto. Estas podrían ser, por ejemplo:

  - ⇨ Tener en cuenta las sentencias *when* del dominio.
  - ⇨ Decidir no tener en cuenta el signo a la hora de identificar los predicados meta en los efectos de las acciones. Es decir, si la meta de un problema tuviese un predicado acompañado de un *not*, buscar ese predicado acompañado, o no, del *not* en los efectos de las acciones.
  - ⇨ En vez de escoger la opción de unirlas al principio (es decir, unir en primer lugar todos los predicados de las acciones con predicados metas distintos en los

efectos), probar uniéndolas al final. O también probar realizando la intersección de conjuntos en lugar de la unión.
- ⇨ Experimentar con los conjuntos *B, C* y *D* descartados.
- ⇨ Experimentar con otros operadores del conjunto, por ejemplo, *UA* (unión de los elementos del conjunto *A*). El que se eligió fue ∩*A*, es decir, una interesección entre todos los predicados ubicados en las precondiciones de las acciones que poseen predicados meta en sus efectos.
- ⇨ Bajar el *improvement* en las PDBs que tengan variables *no relevantes* en el proceso de generación de patrones de *iPDB*. También podría probarse el hecho de bajarlo en variables *no relevantes* y subirlo al analizar variables *relevantes*.
- ⇨ Modificar el factor que incentiva el *impromevent* que las PDBs con variables *relevantes* en el proceso de generación de patrones de *iPDB*.
- ⇨ Modificar *iPDB* para que solo analice variables relevantes a la hora de ir construyendo los patrones.

- Tomar otras alternativas en el desarrollo de la heurística independiente del dominio completamente distintas a las tomadas en el proyecto. Estas podrían ser, por ejemplo:

- ⇨ Aportar otras alternativas acerca de cómo construir una heurística independiente del dominio, ya sea mediante un estudio de las PDBs que algoritmos como *iPDB* o de forma completamente distinta como en *Merge and Shrink*.
- ⇨ Implementar algoritmos más sofisticados que generalicen basándose en las observaciones de las PDBs creadas mediante algún algoritmo de generación de patrones, pero especifiquen hasta el punto de poder hacer distinciones para saber qué variables coger y qué PDBs formar por las características del problema en cuestión, y no solo a las características del dominio. De esta forma, en los dominios estudiados probablemente ese supuesto algoritmo sea capaz de resolver más problemas.
- ⇨ A la hora de obtener una operación entre conjuntos que se ajuste a las variables *relevantes* obtenidas por algoritmos como *iPDB*, emplear programación genética. De esta forma, usando una función de *fitness* relacionada con el parecido entre las variables *relevantes* y las obtenidas por la operación entre conjuntos, se obtendrá una operación entre conjuntos adecuada. En esta investigación podrían estudiarse aspectos relacionados con teoría de conjuntos.
- ⇨ Un trabajo futuro podría consistir en crear un algoritmo que considere también el tipo de los objetos que acompañan a los predicados, tanto al recopilar los predicados *relevantes* (observando aquellos tipos de objetos de los predicados meta), como al seleccionar aquellas variables *relevantes* dentro del *output.sas*. De esta forma, se filtrarían los predicados y variables que no vengan acompañados de un objeto de un tipo apropiado en función a las observaciones previas realizadas.

- ⇨ Codificar un algoritmo genético independiente del dominio que, a partir de un conjunto de variables *relevantes* sea capaz de construir un conjunto de PDBs apropiado.
- ⇨ Codificar un algoritmo independiente del dominio que, además de identificar las variables *relevantes*, sepa construir las PDBs. Entre todas las ideas de futuros proyectos expuestas, esta es en la que más confío para obtener buenos resultados. Realizar un estudio sobre esto es digno de otro Trabajo Fin de Grado. En caso de que alguien esté dispuesto a afrontar este problema, podrá hacer uso de las siguientes observaciones que se han realizado a medida que se llevaba a acabo este actual proyecto:
    - En primer lugar, identificar las variables relevantes es necesario para poder construir las PDBs.
    - Por cada variable meta, sería una buena opción construir PDBs individuales con cada una de ellas, tal y como hace iPDB.
    - Intentar introducir en una sola PDB todas las variables meta que no se dan en el estado inicial. Si estas cupiesen, intentar introducir en esta PDB también todas las variables no meta *relevantes*.
    - En el caso de que no cupiesen, crear varias PDBs, todas con variables no meta *relevantes*, y tantas variables meta como se pueda (siempre que estas no se den en el estado inicial del problema en cuestión), elegidas de forma aleatoria. Esto se haría hasta el punto de que, al menos, en una PDB no individual se encuentre una variable meta que no se dé en el estado inicial.
    - En caso opcional, podría probarse también crear una PDB por cada variable meta, en las que se introduzcan todas las variables no meta *relevantes* y una variable meta.

  Quizás sea necesario recopilar más información de la que se ha dado y/o descartar algunas opciones. Todo dependerá de lo que tenga en consideración el futuro realizador de este proyecto.

- Y, por último, un posible proyecto futuro podría estar relacionado con reimplementar los algoritmos aquí expuestos, optimizando la ejecución, por ejemplo, mediante el uso de *threads*. Sin embargo, personalmente, no lo considero como una gran idea, ya que lo relevante es la potencia del algoritmo, no de su implementación en sí. El producto resultante de este proyecto no son los algoritmos implementados, sino la información recabada. En este trabajo se ha comparado la potencia del algoritmo, no la del código fuente, sobre un mismo nivel. Por otra parte, esto ayudaría a reducir tiempo, lo cual siempre es recomendable en un proyecto tanto práctico como teórico.

SECCIÓN 9. ANEXO

9.1 Abstract

9.1.1 Introduction

The technological advance is being abysmal in this decade. Devices, technologies, algorithms and methods that triumphed a few years ago are now obsolete, and it is not surprising, because we are experiencing a great boom in digitalization. To get here, a lot of people have invested hours in researching new algorithms and new tools to improve our quality of life. As a student of computer science, what attracts me most about this science are the theoretical bases. I would want to contribute a little bit to improve the current technology.

I had the opportunity to do a final project about a research project. Specifically, in automated planning. For this, *Fast Downward* planner and some domains and problems of IPC, The *International Planning Competition*, have been used. The initial idea was to develop a heuristic that operated in a concrete domain using PDBs, that is, *Pattern Databases*; and to be able, subsequently, to generalize by programming a domain-independent heuristic based on the study conducted on these IPC domains. For this, it is necessary to analyse which are the variables and the patterns involved in the resolution of each of these domains, resolving them previously with a planner that use PDBs. After that, to try to generalize so that, given a new domain, automatically identify which variables could be useful and solve most of the problems of that domain in the shortest time and memory usage possible.

Fast Downward provides algorithms capable of generating PDBs and solving problems, but they need to do search previously. These searches usually involve a considerable spending of memory and time. The current research was born because it is necessary to obtain a set of patterns as good or better than the one provided by this type of search algorithms, but in a more guided and quick way. However, the study is limited, and focuses exclusively on obtaining optimal plans, not merely satisfiable.

In this way, if someone wishes to obtain an optimal plan from an initial state to reach a final state through a set of actions, if using PDBs is among their ideas, the information obtained in this project could be very useful.

9.1.2 Motivation and objectives

The raison d'être of this research, as it has been written in the introduction, is to be able to solve planning problems in the shortest time and amount of memory possible, by

studying the variables and patterns involved in the resolution of certain domains, to be able to generalize without being necessary to study the domain. These objectives are listed below in a more formal and complete way:

1. Conduct a study of the state of the art in accordance with the purposes of this project.
2. Address the regulatory framework and apply it to the project.
3. Plan the project, establishing an initial budget and the socio-economic impact that it has.
4. Write requirements, *use cases* and all these aspects related to the design of the solution.
5. Identify which variables are involved in the resolution of planning problems and understand how to group them in PDBs.
6. Program an algorithm that directly constructs the PDBs, test it and accepting it if it becomes as good or better than the best analysed algorithm of *Fast Downward* that resolves them (using the same domain and all the optimization problems of that domain). Draw conclusions based on these results. Carry out this study with 5 domains: *Snake*, *Termes*, *Hiking*, *Spider* and *Tetris*.
7. From the information extracted in each of these domains, conduct a study to be able to obtain these variables and relevant PDBs in the resolution of a planning problem given a domain and any problem, without having to study and analysed it previously.
8. Program an algorithm based on the extracted information, test it and draw conclusions.
9. Present future study lines related to the current project and its results.
10. Write a report that collects all this information (adding the final planning and budget) and exposes it in a clear, clean and understandable way; complying with all the criteria of the correction matrix.

9.1.3 Skeleton of the investigation

Although the research fits to the results obtained during the experimentation, its skeleton was previously defined. This consists on the following steps:

1) **Establishment of the key objectives of the research**: although initially the project objectives were established, it is necessary to emphasize and internalize those objectives referred to the research, to establish an adequate methodology of study.
2) **General questions**: before beginning the experimentation process, it is necessary to specify certain general aspects of the project, such as the programming language, the operating system, etc.

3) **Choice of domains**: choose the domains that will be studied to perform a domain-dependent heuristic for each one of them.
4) **Choice of study algorithms**: analyse the algorithms related to PDBs that *Fast Downward* offers. To establish which of them is necessary to use to obtain the *relevant* variables and to program the algorithm, and which ones are necessary to use for the comparison and evaluation of the programmed algorithm.
5) **Execution of the algorithms and observation of results**: execute the selected algorithms and observe which variables are *relevant* and how they make up the PDBs. Do this for each of the domains chosen in step 3.
6) **Programming and execution of the domain dependent algorithm**: program and execute an algorithm that adjusts to the observations made in the previous step. Do this for each of the selected domains.
7) **Testing the domain-dependent algorithm**: verify its correct performance and validate it if it complies the user requirements.
8) **Evaluation of the algorithm (domain dependent)**: compare the algorithm programmed in the previous step with others based on PDBs. Do this for each of the selected domains.
9) **Extraction of conclusions from the domain**: draw conclusions based on the comparison made in the previous step. Do this for each of the selected domains.
10) **Observation of similar characteristics among the relevant PDBs of each domain**: to study which characteristics all the variables selected as *relevant* in each domain during the investigation share, according to their position in the codification of these.
11) **Programming and / or modification and execution of the domain independent algorithm**: program an algorithm that makes use of the observations made in the previous step, modifying other algorithms if it was necessary. Execute finally all the programmed and / or modified algorithms.
12) **Testing the domain-independent algorithm**: verify its correct performance and validate it if it complies the user requirements.
13) **Evaluation of the algorithm (domain independent)**: compare the algorithm programmed in the previous step with others based on PDBs.
14) **Extraction of conclusions from the domain-independent study**: draw conclusions based on the comparison made in the previous step.
15) **Establishment of possible future works**: based on the conclusions drawn during the realization of this project, establish possible future lines of study that could provide good results, to progress the research initiated here.

9.1.4 Research

9.1.4.1. Previous study

First, it is necessary to perform a preliminary analysis. After this, the domains *Snake*, *Termes*, *Hiking*, *Spider* and *Tetris* will be selected. Regarding the algorithms, *iPDB*, the *Canonical PDB combo* and the *Zero-One PDB combo* will be chosen.

9.1.4.2. *Snake*

After contemplating the PDBs of each problem, it could be possible to generalize. For this, the solved problems will be observed in more detail. The *blocked* variables appear when the problems are simple, but those that always or almost always tend to come out are variables of goals, the *headsnake* and the *ispawn*.

It has been thought, therefore, to program an algorithm that creates PDBs with the following characteristics:

First, build around 9 long PDBs, approximately the number of long PDBs that *iPDB* finally offers. All have the *headsnake* variable, the *ispawn* variable and random food points of the goal. The number of variables per PDB is approximately 14, also according to the PDBs that *iPDB* displays in its algorithm. In addition, all the goal variables should be among all the patterns.

Also, put all the initial food points as PDBs of a single variable. First place the largest patterns, and then the individual patterns, to favour *Zero-One PDB*, although for *Canonical PDB* the ordering is not relevant.

After programming the algorithm, executing it and comparing it with the rest, several conclusions can be reached regarding this domain:

When the problems are extremely simple, it will be enough to use one of the *combo* algorithms, especially *Zero-One*, to spare the process of generating a slightly more informed heuristic. However, when the complexity starts to rise more, using *AlgSnake* with *Canonical* is the best option. *iPDB* builds better heuristics in simple or intermediate difficult problems, but *AlgSnake* in *Canonical*, because it generalizes, is more powerful in complex problems. Anyway, this small difference that *iPDB* gains in these problems is lost by using excessive time to generate the set of patterns.

In short, the best algorithm is the one that has been programmed, that is, *AlgSnake* through *Canonical*. It solves more problems, and in simple problems there is not a big temporal difference respect to the other algorithms.

9.1.4.3. *Termes*

After contemplating the PDBs of each problem, it is difficult to generalize, because the number of variables that exist is very small, and many of them are goal variables. For that reason, in addition to observing those problems solved in more detail, logic and common sense will be used to program the algorithm.

First, it is not possible to incorporate all the variables in a unique PDB, because if it could be done, the *combo* algorithm would show better results than it shows. That is why it is necessary to know how to choose quite well which variables related to the heights incorporate. That said, the best alternative is to build three types of PDBs:

- A PDB with the variable that indicates the position of the robot and each of the variables that refers to positions in which the goal and the initial state have different heights. These variables will be called, from now on, *variable heights*.

- A set of PDBs resulting from all the combinations between all the adjacent positions (excluding the quarry) at the variable heights, also adding the *hasblock* variable and the position of the robot to each of them. For example, if a problem had two variable heights (*H1* and H2), and one of them had 3 neighbours (*A*, *B* and *C*) and the other only two (*X* and *Z*), (none of them is the position of the quarry) the number of PDBs would be 6: [*H1*, *H2*, *A*, *X*, *hasblock* and the robot position], [*H1*, *H2*, *A*, *Z*, *hasblock* and the robot position], [H1, H2, B, X, hasblock and the robot position], [*H1*, *H2*, *B*, *Z*, *hasblock* and the robot position], [*H1*, *H2*, *C*, *X*, *hasblock* and the robot position], [*H1*, *H2* , *C*, *Z*, *hasblock* and the position of the robot]. This is reasonable because, if we want that the robot builds in a position, it is necessary to visit one of its adjacent positions. As is evident, all the repeated variables are suppressed. In addition, PDBs greater than 10 variables will be discarded.

To consider also the neighbours of the neighbours of the *variable heights* gives a big and unnecessary combinatorial explosion.

- A long PDB with all the variables that have appeared in the PDBs named above, if the number of variables doesn't exceed a certain value, to not get an error message. That number was set at 10.

In addition, all the variables related to the height will be put in an individual PDB, except those that represent *variable heights* because they appear in all the PDBs. This action forces all variables to appear in at least one PDB. First the largest patterns are set, and then the individual patterns, to favour *Zero-One PDB*, although for *Canonical PDB* the ordering is not relevant.

After programming the algorithm, executing it and comparing it with the rest, it is difficult to draw clear conclusions about which is the best algorithm.

Firstly, what is clear is that neither *AlgTermes* in *Zero-One* and *Zero-One* using *combo* can compete against the algorithm programmed in *Canonical* or against *iPDB*. Then, according to the memory used, the two remaining algorithms compete very well. Regarding time, *AlgTermes* in *Canonical* provides better timestamps more frequently, also in the more complex problems. On the other hand, as it is evident, the heuristic of *iPDB* when it incorporates almost all the variables in a PDB is more informed, but due to the construction time of these, it is not worth it. In problems in which the number of *variable heights* is small, it seems to indicate that *iPDB* also leads, because it is able to solve a problem more than the rest, although this means a higher consumption of time. However, in problems with more *variable heights*, *AlgTermes* in *Canonical* takes the lead.

iPDB is appropriate, therefore, when you want to solve a problem with few variable goals (its search using *Hill climbing* is usually very appropriate to choose the variables that can work best), and *AlgTermes* otherwise (due to the combinatorial of neighbouring positions). However, it is advisable to execute always in *AlgTermes* before running on *iPDB*, to improve the timestamps (although it is not sure that it can be solved) because it usually provides shorter times. In the same way, it is possible to achieve better times if it is executed in *AlgTermes* after executing in *iPDB*. Choosing one of the two algorithms is difficult, because it depends on the problem itself; but if I had to choose one I would opt for *AlgTermes* using *Canonical*, because *iPDB* does not stand out on this in any of the previous tables (except in the table of the number of problems solved). *AlgTermes* in *Canonical* has generally provided better results.

9.1.4.4. *Hiking*

In this domain it is advisable to include multiple variables in a single PDB (due to the good results of the *combo* algorithm). The most logical thing is to create PDBs based on the observations of the first problems, that is, consider the variables of the tent. That is why, by collecting all this information, we can obtain this set of PDBs:

- An individual pattern for each *walked* variable.

- A long pattern for each *walked* variable that has the problem, together with each member of the pair related to this walked variable, and all the variables related to the tents, that is, the position and if it is assembled or disassembled.

First the largest patterns are set, and then the individual patterns, to favour *Zero-One PDB*, although for *Canonical PDB* the ordering is not relevant. The goals correspond to the *walked* variables, so they always appear in the set of PDBs created.

After programming the algorithm, executing it and comparing it with the rest, a clear conclusion can be reach regarding this domain. It is easy to conclude that the study of the variables has given very good results. A very powerful algorithm has been created. To affirm that *AlgHiking* in *Canonical* is the best of the 5 algorithms is a safe bet. I refer to the results obtained above.

9.1.4.5. *Spider*

As an initial note, it is necessary to indicate that the set of PDBs obtained after the pruning has been considered, but not the additive sets, because these are created by *Canonical* directly. Having said that, it has been decided to create the following PDBs:

- A small PDB for each goal, because there is not a clear and simple pattern to prune those that the *Canonical* algorithm will discard them. There are also certain peculiarities. The variable *clear deal* 1 appears when there is more than one packet of cards (that is, there is a card that is written as follows: *d1-sx-vy*, being *x* and *y* natural numbers), but what we want to do is to generalize and apply the same algorithm to build the PDBs in the same way. Do not choose between algorithms depending on the characteristics of the problem. This could be considered in some other alternative project.

- Because 80% of the time the variable cur*rently-collecting-deck* with some card from the *pile* has come out, making all the combinations of these two values could be a good idea.

- Most of the cards that appear with *current-deal* and *currently-dealing* are cards in *deal* and *clear deal* variables. That said, a good alternative is to do the whole combination of *current-deal* and *currently-dealing* with the *deal cards* (*on* variables) and the *clear deal* (creating PDBs of 3 variables), and the whole combination of *current-deal* and *current-dealing* with *clear deal* and *deal* cards at the same time (creating PDBs with 4 variables).

In this way, all the goal variables should be among all the patterns. In addition, all PDBs are ordered by the size (number of variables), to favour *Zero-One PDB*, although for *Canonical PDB* the ordering is not relevant.

When the algorithm is programmed, executed it and compared with the rest of algorithms, several conclusions can be reached regarding this domain:

In case of choosing between the algorithm *AlgSpider* using *Canonical* or using *Zero-One*, because they solve the same number of problems, the best alternative would be *Zero-One*. Even though its heuristic is worse and uses more memory, the difference is not as significant. It is worth sacrificing the quality of the heuristic a little bit to get quite smaller times.

Regarding the comparison between *AlgSpider* in *Zero-One* and *iPDB*, it is difficult to establish a better algorithm. The positive point of *AlgSpider* is the total time and the memory used, that they are even less compared to *iPDB*. But, on the other hand, *iPDB* elaborates a better heuristic in most of the problems (although the difference is not very meaningful), and it is an algorithm that knows how to adapt well to the conditions of the problem, getting to solve two more problems than *AlgSpider* in *Zero-One*.

It is much more difficult to generalize in this domain. To solve more problems than *iPDB* it is necessary to distinguish between these to consider applying an algorithm or another one. It is not completely a generalization and it would complicate the algorithm at the expense of getting better results.

If I had to choose one of the two algorithms, I would opt for *AlgSpider* in *Zero-One*, for the simple fact that it solves the problems in much less time, and the number of additional problems solved by *iPDB* are only 2. A good practice would be to run a problem in *AlgSpider* using *Zero-One*, and if it is not solved, opt for *iPDB*, which is better suited to the conditions of the problem. *AlgSpider* is equated to the power of *iPDB*, so it meets the requirements to be accepted.

9.1.4.6. *Tetris*

The *clear* variables predominate, both those related to *meta* predicates and those that are not, when they are in consecutive order. *iPDB* does not provide much more information. However, *combo* shows that it is important to create extensive PDBs, with 19 variables at most. Nevertheless, this is not enough to create an algorithm that pretends to overcome the temporary marks of the *combo* algorithm.

Observing the domain itself, you can see how the *Tetris* pieces move horizontally and vertically. That said, the programmed algorithm must have the following characteristics to generalize:

- A PDB that acts as the *combo* algorithm, that is, that gathers 19 *clear* variables. However, because the pieces can be moved horizontally, variables will be taken in an orderly manner starting from the upper left corner, with the purpose of ending in the lower right corner of the strip related to the meta clearings (moving all pieces of the top of the board to the lower positions). If all meta variables are included in a single PDB,

this will be completed with *clear* non-meta variables until reaching the 19 variables, following the same order. However, if there are still meta *clear* variables remaining when 19 variables are set in the PDB, another new PDB with 19 variables will be introduced, but starting with the third last variable of this completed PDB (to set something in common between PDBs) and following the same process. In this way, we make sure that all the goal patterns are in a PDB at least.

- Because that the pieces in the Tetris falls, it could be interesting to create a PDB per column, inserting all the *clear* variables of the positions of that same column in it. In this way, all *clear* variables will be added in one PDB at least, and every PDB will have one meta variable at least.

Individual PDBs (with only one variable) will not be considered because all the positions are linked together, and all these variables joined together provide more information than separately. These individual PDBs would not contribute to the heuristic. In addition, all patterns have been ordered from highest to lowest to favour *Zero-One PDB*, although for *Canonical PDB* the ordering is not relevant.

When the algorithm is programmed, executed it and compared with the rest of algorithms, a conclusion can be reached regarding this domain. In this case, it is not difficult to choose an algorithm. *AlgTetris* is superior to *iPDB* and *combo* algorithms. However, in case of choosing between *AlgTetris* in *Canonical* and between *Zero-One*, due to the quality of the heuristic *Canonical* option would be better, although the improvement difference was so small.

9.1.4.7. Domain independent

The domain independent study follows a progressive reasoning, namely, decisions are based on the results. The final decision that was reached was that it could be a good idea to create an algorithm that obtained a set of variables to be used by *iPDB*. In this way, *iPDB* would encourage the PDBs that contain the variables obtained by the previous algorithm, increasing their *improvement* value. These variables are obtained by locating the predicates of the actions that originate a change in a meta predicate (considering the sign of the predicate in the problem definition) and making a disjoint between all the sets of predicates obtained. After this, the relevant variables are those that have an instance of some of the predicates obtained after the disjoint.

When the algorithm is programmed, executed it and compared with the rest of algorithms, several conclusions can be reached:

Firstly, algorithms that create a domain-dependent heuristic are better than modified *iPDB*. Also, this last algorithm is like *iPDB*. Empirically, it has been shown that it would

not be a good alternative to put this modified algorithm in general terms against any of the other 5 programmed. On the other hand, in case of not carrying out the research process to program the algorithm dependent on the domain, the use of *iPDB* or its modified version lies in the characteristics of the domain and the problem to be solved, not in the power of these two algorithms. Basically, if the appearance of *relevant* variables favours the resolution of the domain and the problem, this modified algorithm may be the best alternative. Choosing one of these two as best is impossible. That is why one alternative would be to run the domain dependent algorithm. If this does not solve the problem, run one of the *iPDB* (modified or not). If it still does not solve it, run the other, and ultimately run the algorithms *combo*.

9.1.5 General conclusions

The **first** conclusion shows that the best algorithms (used in this project) are those programmed related to a domain-dependent heuristic (through *Canonical*). Either for the time saved in the generation of patterns, in the worst case, or for the quality of the obtained heuristics. We can affirm the quality of the heuristics provided by this algorithm because the domains that have been chosen are very different from each other. These algorithms are followed by any of the iPDBs (*Fast Downward* or the *modified* one for the domain-independent study). However, its effectiveness depends on the characteristics of the problem and the domain itself. Empirically, we can see that the interpretation of *Canonical's* heuristic is better than the *Zero-one* heuristic interpretation (in general terms).

That said, a good work methodology could be execute some *combo* algorithm (*Canonical* or *Zero-One*) on a problem that you want to solve, because they work in a better way with extremely simple problems. If this does not solve the problem, run *iPDB* and / or *iPDB* modified. If it has not yet been resolved, conduct a study of the relevant variables of the domain, program a domain-dependent algorithm and execute it with *Canonical PDB*. And, ultimately, in case *Canonical PDB* shows results that are much worse than *iPDB*, try to run the domain-dependent algorithm using *Zero-One*. However, there are still algorithms related to PDB to analyse, such as the genetic algorithms in *Canonical* and in *Zero-One*. This can be the basis of future research.

An explanation of the success of these programmed algorithms is that, in addition to the time saved in the construction of the patterns, the research carried out focuses on identifying similar characteristics in the formation of PDBs. In this way, by generalizing, you can achieve patterns that *iPDB* aspired to but did not achieve due to the way it worked and stopped the process of obtaining of PDBs. On the other hand, due to this generalization, some executions on some problems got worse results than *iPDB*,

because *iPDB* fits better to the specific characteristics of the domain and the problem itself.

To compare the power of these programmed algorithms, it is important to mention that both *AlgSnake* in *Canonical* and *AlgTetris* in *Canonical* solve more problems than the best planners used in IPC competitions. *AlgSnake* in *Canonical* solves 16 problems, and the best planner of the competition, 14. On the other hand, *AlgTetris* solves 12, and the biggest number of problems solved by a planner in the competition was 11.

The **second** conclusion answers the question of whether it is worth conducting a study of the PDBs provided by algorithms such as *iPDB* to program an algorithm. The answer is a resounding yes. Although it depends on the problem itself. It is essential to know if it is profitable to invest time in the development of an algorithm that provides domain-dependent heuristics in a specific domain, observing the number of problems to be solved or how well the algorithms already programmed work on these domains and problems. The quality of the programmed algorithm is intimately related to the precision of the research. Also, it is not so necessary to generalize much, as long as it adapts to the domain and the problems. In future projects, it would be nice to compare more than one domain-dependent algorithm with each other. On the other hand, including other algorithms in the observation process could also be a great idea.

The **third** conclusion is related to the term of *relevant* predicates. The variables that work well in PDBs are those related to the predicates that appear in the preconditions of all the actions whose effects contain a predicate related to the goals of the problem to solve. However, this is not generalizable and can be denied based on the results of future research that attend this topic.

The **fourth** conclusion answers the question of whether it is worth investigating domain-independent heuristics. Firstly, even though the two *iPDB* algorithms are similar, one stands out above the other on certain occasions. While it is true that the research conducted has not resulted in a powerful algorithm, the fact that it stands out above *iPDB* in some cases is a clear indication that it would be a good idea to spend more time developing a domain-independent heuristic through more studies. During the research carried out, possible future projects have been discussed. In short, it has been shown that the results of *iPDB* can be improved by developing a domain-independent heuristic.

The **fifth** and last conclusion is required to classify the project carried out. Observing the large number of proposed future line of works and the good results obtained, this project could be classified as a *base*. All future research based on these results and proposals should aim to improve, deepen and / or disprove what has been discussed here. Basically, the quality of this project lies, in great measure, in the proposals drafted during this project. The socio-economic impact of this study and the proposed research is considerable, because of the advance of the technology of today.

It should be noted that all these investigations have been conducted on domains that worked well with PDBs. Otherwise, it does not mean that the conclusions reached here were not applicable to these domains. It would be necessary to carry out experiments on them before issuing an objective judgment.

9.2 Prueba de los algoritmos

9.2.1 *Snake*

- El algoritmo implementado sobre el problema 10 muestra las siguientes PDBs:

[100, 130, 131, 132, 134, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[100, 130, 131, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[100, 130, 131, 133, 134, 135, 136, 137, 138, 139, 140, 141, 142, 143]

[100, 130, 132, 133, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[100, 130, 131, 132, 133, 135, 136, 137, 138, 139, 140, 142, 143, 144]

[100, 130, 131, 132, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[100, 130, 131, 133, 135, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[100, 130, 133, 134, 136, 137, 138, 139, 140, 141, 142, 143, 144]

[131]

[132]

[133]

[134]

[135]

100: *ispawn* (siguiente punto de comida que aparecerá).

130: cabeza de la serpiente.

131 – 135: puntos de comida iniciales.

136 – 144: el resto de los puntos de comida.

- El algoritmo implementado sobre el problema 20 muestra las siguientes PDBs:

[142, 186, 187, 189, 190, 192, 193, 194, 195, 196, 197, 198, 199, 200]

[142, 186, 187, 190, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200]

[142, 186, 188, 189, 190, 191, 192, 193, 194, 195, 196, 197, 198, 200]

[142, 186, 187, 188, 190, 192, 193, 194, 195, 196, 197, 198, 199, 200]

[142, 186, 188, 189, 190, 192, 193, 194, 195, 196, 197, 198, 199, 200]

[142, 186, 189, 191, 192, 193, 194, 195, 196, 197, 198, 199, 200]

[187]

[188]

[189]

[190]

[191]

142: *ispawn* (siguiente punto de comida que aparecerá).

186: cabeza de la serpiente.

187 – 191: puntos de comida iniciales.

192 – 200: el resto de los puntos de comida.

9.2.2 *Termes*

- El algoritmo implementado sobre el problema 10 muestra las siguientes PDBs:

[0, 2, 4, 6, 8, 11, 12]

[0, 4, 6, 8, 11, 12]

[0, 2, 4, 6, 8, 12]

[0, 4, 6, 11, 12]

[0, 2, 4, 6, 12]

[0, 4, 6, 8, 12]

[0, 4, 6, 12]

[0, 4, 6]

[11]

[10]

[9]

[8]

[7]

[5]

[3]

[2]

[1]

0: la posición del robot.

12: *hasblock*.

1 – 11: posiciones del tablero.

4, 6: *alturas variables*.

- El algoritmo implementado sobre el problema 12 muestra las siguientes PDBs:

[0, 1, 3, 4, 6, 7, 10, 11, 12]

[0, 1, 3, 4, 6, 9, 10, 11, 12]

[0, 1, 2, 3, 4, 6, 10, 11, 12]

[0, 1, 2, 3, 4, 6, 7, 11, 12]

[0, 1, 2, 3, 4, 6, 9, 11, 12]

[0, 1, 3, 4, 6, 10, 11, 12]

[0, 1, 3, 4, 6, 7, 11, 12]

[0, 1, 3, 4, 6, 9, 11, 12]

[0, 1, 2, 3, 4, 6, 11, 12]

[0, 1, 4, 6, 7, 10, 11, 12]

[0, 1, 4, 6, 9, 10, 11, 12]

[0, 1, 2, 4, 6, 10, 11, 12]

[0, 1, 2, 4, 6, 7, 11, 12]

[0, 1, 2, 4, 6, 9, 11, 12]

[0, 1, 3, 4, 6, 11, 12]

[0, 1, 4, 6, 10, 11, 12]

[0, 1, 4, 6, 7, 11, 12]

[0, 1, 4, 6, 9, 11, 12]

[0, 1, 2, 4, 6, 11, 12]

[0, 1, 4, 6, 11, 12]

[0, 1, 4, 6, 11]

[10]

[9]

[8]

[7]

[5]

[3]

[2]

0: la posición del robot.

12: *hasblock*.

1 – 11: posiciones del tablero.

1, 4, 6 y 11: *alturas variables*.

9.2.3 *Hiking*

- El algoritmo implementado sobre el problema 7 muestra las siguientes PDBs:

[0, 1, 2, 3, 4, 6, 10]

[0, 1, 2, 3, 7, 9, 11]

[11]

[10]

0, 1, 2 y 3: variables relacionadas con las dos tiendas de campaña.

4, 6: variables relacionadas con la posición de los miembros de una de las parejas.

7, 9: variables relacionadas con la posición de los miembros de la otra pareja.

10 y 11: variables *walked*.

- El algoritmo implementado sobre el problema 20 muestra las siguientes PDBs:

[0, 1, 6, 7, 8, 10, 12]

[0, 1, 6, 9, 10, 11, 13]

[13]

[12]

0, 1, 6 y 10: variables relacionadas con las dos tiendas de campaña.

7, 8: variables relacionadas con la posición de los miembros de una de las parejas.

9, 11: variables relacionadas con la posición de los miembros de la otra pareja.

12 y 13: variables *walked*.

9.2.4 *Spider*

- El algoritmo implementado sobre el problema 10 muestra las siguientes PDBs:

[0, 1, 362, 364]

[0, 1, 362, 365]

[0, 1, 362, 366]

[0, 1, 362, 367]

[0, 1, 362, 368]

[0, 1, 362, 369]

[0, 1, 362, 370]

[0, 1, 362, 371]

[0, 1, 363, 364]

[0, 1, 363, 365]

[0, 1, 363, 366]

[0, 1, 363, 367]

[0, 1, 363, 368]

[0, 1, 363, 369]

[0, 1, 363, 370]

[0, 1, 363, 371]

[0, 1, 362]

[0, 1, 363]

[0, 1, 364]

[0, 1, 365]

[0, 1, 366]

[0, 1, 367]

[0, 1, 368]

[0, 1, 369]

[0, 1, 370]

[0, 1, 371]

[249, 346]

[249, 347]

[249, 348]

[249, 349]

[249, 350]

[249, 351]

[249, 352]

[249, 353]

[249, 354]

[249, 355]

[249, 356]

[249, 357]

[249, 358]

[249, 359]

[249, 360]

[249, 361]

[375]

[374]

[373]

[372]

[371]

[370]

[369]

[368]

[367]

[366]

[365]

[364]

[363]

[362]

[361]

[360]

[359]

[358]

[357]

[356]

[355]

[354]

[353]

[352]

[351]

[350]

[349]

[348]

[347]

[346]

0: *current-deal*.

1: *currently-dealing*.

249: *currently-collecting-deck*.

362 y 363: los *clear deal*.

346 –361: cartas de *pile*.

364 – 371: cartas de *deal*.

372 – 375: los clear de *pile*.

- El algoritmo implementado sobre el problema 15 muestra las siguientes PDBs:

[0, 1, 211, 213]

[0, 1, 211, 214]

[0, 1, 211, 215]

[0, 1, 211, 216]

[0, 1, 211, 217]

[0, 1, 211, 218]

[0, 1, 212, 213]

[0, 1, 212, 214]

[0, 1, 212, 215]

[0, 1, 212, 216]

[0, 1, 212, 217]

[0, 1, 212, 218]

[0, 1, 211]

[0, 1, 212]

[0, 1, 213]

[0, 1, 214]

[0, 1, 215]

[0, 1, 216]

[0, 1, 217]

[0, 1, 218]

[170, 201]

[170, 202]

[170, 203]

[170, 204]

[170, 205]

[170, 206]

[170, 207]

[170, 208]

[170, 209]

[170, 210]

[221]

[220]

[219]

[218]

[217]

[216]

[215]

[214]

[213]

[212]

[211]

[210]

[209]

[208]

[207]

[206]

[205]

[204]

[203]

[202]

[201]

0: *current-deal*.

1: *currently-dealing*.

170: *currently-collecting-deck*.

211 y 212: los *clear deal*.

201 –210: cartas de *pile*.

213 – 218: cartas de *deal*.

219 – 221: los *clear* de *pile*.

9.2.5 *Tetris*

- El algoritmo implementado sobre el problema 13 muestra las siguientes PDBs:

[1173, 1177, 1178, 1179, 1180, 1181, 1182, 1183, 1184, 1185, 1186, 1187, 1188, 1189, 1190, 1191, 1192, 1193, 1194]

[1163, 1167, 1169, 1173, 1179, 1183, 1185, 1187]

[1165, 1171, 1175, 1177, 1181, 1189, 1191, 1193]

[1166, 1172, 1176, 1178, 1182, 1190, 1192, 1194]

[1164, 1168, 1170, 1174, 1180, 1184, 1186, 1188]

1163 – 1194: variables *clear*.

Cada variable clear está relacionada con una casilla del tablero de juego, a continuación, se muestra a qué posiciones corresponden las anteriores variables. Las casillas con un tono más oscuro hacen referencia a variables meta (*tabla 191*):

1179	1181	1182	1180
1183	1189	1190	1184
1185	1191	1192	1186
1187	1193	1194	1188
1173	1177	1178	1174
1169	1175	1176	1170
1167	1171	1172	1168
1163	1165	1166	1164

Tabla 191. Tablero de juego de la segunda prueba

- El algoritmo implementado sobre el problema 17 muestra las siguientes PDBs:

[1830, 1831, 1832, 1833, 1834, 1835, 1836, 1837, 1838, 1839, 1841, 1842, 1843, 1844, 1845, 1846, 1847, 1848, 1849]

[1810, 1812, 1814, 1815, 1816, 1817, 1818, 1819, 1820, 1821, 1822, 1823, 1824, 1825, 1826, 1827, 1828, 1829, 1840]

[1810, 1814, 1816, 1820, 1824, 1830, 1834, 1836, 1838, 1839]

[1812, 1818, 1822, 1826, 1828, 1832, 1842, 1844, 1846, 1848]

[1813, 1819, 1823, 1827, 1829, 1833, 1843, 1845, 1847, 1849]

[1811, 1815, 1817, 1821, 1825, 1831, 1835, 1837, 1840, 1841]

1810 – 1849: variables *clear*.

Cada variable clear está relacionada con una casilla del tablero de juego, a continuación, se muestra a qué posiciones corresponden las anteriores variables. Las casillas con un tono más oscuro hacen referencia a variables meta (*tabla 192*):

1830	1832	1833	1831
1834	1842	1843	1835
1836	1844	1845	1837
1839	1846	1847	1841
1838	1848	1849	1840
1824	1828	1829	1825
1820	1826	1827	1821
1816	1822	1823	1817
1814	1818	1819	1815
1810	1812	1813	1811

Tabla 192. Tablero de juego de la tercera prueba

9.3 Evaluación de los algoritmos

9.3.1 *Snake*

Tiempo creación PDBs (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSnake (Canonical PDB)*	*AlgSnake (Zero-One PDB)*
P01	897.011	0.833728	0.730376	5.69984393	6,45494993
P02	279.025	3.12795	2.73849	9.78233408	9,06734292
P03	346.773	3.78007	3.57676	11.42436882	11,19791089
P04	84.4693	0.259746	0.340687	0.11052613	0,10134106
P05	-	0.0685216	0.137968	0.59178407	0,53052592
P06	446.251	2.80812	2.89684	13.07605392	13,08660586
P07	300.992	3.48439	3.60363	17.23160521	17,11166788
P08	412.976	4.07835	4.16116	24.31779686	22,50430198
P09	912.14	0.363003	0.381127	0.33561602	0,46899588
P10	430.822	0.450381	0.556201	6.80652390	5,31369780
P11	641.545	0.595919	0.844009	15.48361493	16,23203703
P12	517.429	3.52573	3.68083	23.69320990	23,96003997
P13	691.745	4.01962	4.27944	31.17903194	31,20318815
P14	452.627	4.47418	4.69643	48.48496502	39,94114497
P15	954.81	0.197881	0.437467	2.60191696	2,20183593
P16	705.208	3.13461	3.36671	28.10543607	28,82528503

P17	994.748	4.00988	4.33783	39.39794115	39,92463490
P18	679.166	4.63147	4.91074	56.25695809	57,21681694
P19	896.385	5.05517	5.40392	84.95284791	79,54633187
P20	1173.1	0.897972	1.10336	9.92919088	11,85245495

Tabla 193. Tiempo de creación de PDBs – Snake – Evaluación

Tiempo poda y creación conjuntos aditivos (s)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgSnake (Canonical PDB)***	***AlgSnake (Zero-One PDB)***
P01	0.000114591	3.5258e-05	-	7.9294e-05	-
P02	0.000204172	4.2847e-05	-	0.000114895	-
P03	0.000104696	5.2086e-05	-	8.2989e-05	-
P04	4.6969e-05	2.9518e-05	-	5.1991e-05	-
P05	-	1.5494e-05	-	6.5925e-05	-
P06	8.1162e-05	3.851e-05	-	9.3987e-05	-
P07	7.7994e-05	4.421e-05	-	0.000127157	-
P08	0.000101461	5.5417e-05	-	0.000116475	-
P09	6.336e-05	2.8374e-05	-	5.3749e-05	-
P10	0.000241553	2.9899e-05	-	0.000111777	-
P11	8.4443e-05	3.0247e-05	-	0.000102429	-
P12	0.000100337	4.922e-05	-	8.9804e-05	-

P13	0.000125804	6.6747e-05	-	9.1854e-05	-
P14	0.000130829	9.1435e-05	-	0.000124142	-
P15	7.7087e-05	2.8385e-05	-	4.7566e-05	-
P16	8.8873e-05	4.6806e-05	-	0.000121876	-
P17	0.000104036	7.7531e-05	-	0.000116227	-
P18	0.000130395	8.9724e-05	-	8.8725e-05	-
P19	0.000149052	0.000108824	-	9.7827e-05	-
P20	8.7738e-05	2.8916e-05	-	0.000101114	-

Tabla 194. Tiempo poda y creación conjuntos aditivos – Snake – Evaluación

Tiempo búsqueda (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSnake (Canonical PDB)*	*AlgSnake (Zero-One PDB)*
P01	0.00180985	0.0930085	0.067105	0.00173066	0.00497502
P02	0.0558545	8.32186	7.58065	0.0710127	0.450683
P03	14.5863	46.1717	45.9429	4.36722	24.5387
P04	0.0007224	0.000746461	0.000794169	0.000887084	0.00100648
P05	-	0.00323584	0.00270144	0.00104429	0.00229828
P06	0.010764	11.7874	11.8813	0.0165334	0.768081
P07	212.179	-	-	37.3199	198.575
P08	-	-	-	634.851	-
P09	0.00122594	0.00341853	0.00353939	0.00116513	0.00176191

P10	0.00175552	0.142863	0.141851	0.00196234	0.00223223
P11	0.0067074	31.7953	31.915	0.00694257	0.219762
P12	149.223	-	-	27.4568	465.923
P13	-	-	-	-	-
P14	-	-	-	-	-
P15	0.00179786	0.0755458	0.0758174	0.00177367	0.0107408
P16	4.0889	460.743	460.997	0.799301	33.7912
P17	-	-	-	595.17	-
P18	-	-	-	-	-
P19	-	-	-	-	-
P20	0.00192742	0.430593	0.436789	0.00193004	0.0753995

Tabla 195. Tiempo búsqueda – Snake – Evaluación

Memoria total utilizada (KB)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSnake (Canonical PDB)*	*AlgSnake (Zero-One PDB)*
P01	3966840	16528	16528	49484	47312
P02	1375808	69828	69932	70800	65248
P03	1692756	265144	265136	101756	213080
P04	614976	14404	14144	7412	7412
P05	-	8684	8684	10752	10516
P06	2008968	89368	89368	84240	78896

P07	1282872	-	-	338372	1152200
P08	-	-	-	2898312	-
P09	3456680	18944	18944	9988	10760
P10	1916044	13684	13944	49564	35492
P11	2469328	259240	259312	90648	79056
P12	1869348	-	-	275728	2522076
P13	-	-	-	-	-
P14	-	-	-	-	-
P15	3614664	13220	13220	22448	19564
P16	2264672	2349492	2349516	132312	295384
P17	-	-	-	3007224	-
P18	-	-	-	-	-
P19	-	-	-	-	-
P20	3806524	21084	21084	56764	57724

Tabla 196. Memoria total utilizada – Snake – Evaluación

9.3.2 *Termes*

Tiempo creación PDBs (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgTermes (Canonical PDB)*	*AlgTermes (Zero-One PDB)*
P01	3.56702	0.532203	0.510582	0.00926041	0,01163983
P02	3.49414	0.512825	0.525183	0.01979477	0,01370886
P03	4.28782	0.784493	0.778698	0.02051925	0,01666934
P04	4.41132	0.673734	0.758701	0.14733695	0,14728294
P05	3.08709	0.50107	0.578178	0.01958222	0,01757559
P06	3.07474	0.489759	0.658113	0.42538806	0,46217993
P07	2.35627	0.206082	0.233965	0.01711157	0,00549694
P08	1.53309	0.194055	0.24241	6.62498118	6,65356120
P09	2.57898	0.418065	0.459052	0.04013943	0,03621250
P10	2.60056	0.433163	0.433148	0.40316097	0,41379196
P11	4.92451	0.491494	0.575349	0.71341210	0,76018588
P12	3.39212	0.520355	0.549935	1.00851204	1,11127396
P13	4.47534	0.787527	0.815485	1.13943313	1,15679814
P14	4.33649	0.740431	0.761869	0.25544107	0,19315992
P15	2.98447	0.556789	0.583928	3.18947212	3,55631103
P16	1.7994	0.521232	0.488756	43.21525800	43,83086909
P17	3.73993	0.629801	0.585413	8.51733798	9,50202300

P18	3.02609	0.557013	0.512907	5.35232609	6,00353016
P19	4.68182	0.822789	0.715535	41.68896885	44,37997396
P20	2.68327	0.878545	0.841016	7.17378813	6,90845484

Tabla 197. Tiempo de creación de PDBs – Termes – Evaluación

Tiempo poda y creación conjuntos aditivos (s)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTermes (Canonical PDB)***	***AlgTermes (Zero-One PDB)***
P01	4.9804e-05	3.0748e-05	-	5.8889e-05	-
P02	6.1458e-05	3.4431e-05	-	6.7321e-05	-
P03	7.8074e-05	3.3395e-05	-	4.7878e-05	-
P04	6.7799e-05	2.551e-05	-	5.5715e-05	-
P05	5.826e-05	4.0292e-05	-	5.1274e-05	-
P06	4.5084e-05	4.5271e-05	-	5.4754e-05	-
P07	5.0743e-05	4.1889e-05	-	0.000137399	-
P08	5.7357e-05	4.1541e-05	-	5.6914e-05	-
P09	4.2588e-05	3.9385e-05	-	0.00013703	-
P10	5.026e-05	4.0414e-05	-	5.3525e-05	-
P11	5.2188e-05	4.1184e-05	-	5.2686e-05	-
P12	4.4521e-05	3.1934e-05	-	7.4213e-05	-
P13	4.2446e-05	4.6666e-05	-	0.000111646	-

P14	4.0861e-05	3.7556e-05	-	5.7211e-05	-
P15	4.1828e-05	4.262e-05	-	5.0921e-05	-
P16	3.7237e-05	3.8975e-05	-	8.0752e-05	-
P17	4.6544e-05	2.2184e-05	-	0.000139167	-
P18	4.7081e-05	3.3608e-05	-	0.000106998	-
P19	3.9261e-05	2.4661e-05	-	0.000125596	-
P20	4.0351e-05	2.4846e-05	-	6.0391e-05	-

Tabla 198. Tiempo poda y creación conjuntos aditivos – Termes – Evaluación

Tiempo búsqueda (s)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTermes (Canonical PDB)***	***AlgTermes (Zero-One PDB)***
P01	0.0215248	0.159746	0.161164	0.102681	0.10197
P02	0.0749325	3.2648	3.29703	0.604941	0.561246
P03	0.564836	16.2323	16.185	6.06744	6.25571
P04	2.9279	34.1842	33.6204	5.41609	5.45299
P05	268.81	-	-	-	-
P06	-	-	-	-	-
P07	-	-	-	-	-
P08	-	-	-	-	-
P09	-	-	-	-	-
P10	-	-	-	-	-

P11	0.820775	10.4601	10.2966	0.51604	0.546264
P12	0.202634	1.76319	1.72309	0.269315	0.37535
P13	53.6902	77.6181	76.6403	52.3036	75.7406
P14	7.35762	235.312	228.613	35.357	37.1941
P15	-	-	-	-	-
P16	-	-	-	-	-
P17	78.0712	188.248	144.847	91.8268	133.9
P18	12.5837	46.0834	32.1908	9.2814	18.4567
P19	22.3321	90.2797	74.0481	16.312	34.8082
P20	124.186	327.523	242.325	66.6737	114.912

Tabla 199. Tiempo búsqueda – Termes – Evaluación

Memoria total utilizada (KB)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTermes (Canonical PDB)***	***AlgTermes (Zero-One PDB)***
P01	49240	9660	9664	6104	6048
P02	49080	52408	52428	11744	11824
P03	61760	224692	224940	102988	102768
P04	63816	443452	443428	103692	103912
P05	3288164	-	-	-	-
P06	-	-	-	-	-

P07	-	-	-	-	-
P08	-	-	-	-	-
P09	-	-	-	-	-
P10	-	-	-	-	-
P11	67124	186164	186172	21188	19404
P12	49072	29812	29836	16796	22140
P13	766876	977964	978068	775904	898484
P14	114188	3087500	3088432	449900	450688
P15	-	-	-	-	-
P16	-	-	-	-	-
P17	847824	1642596	1642692	929328	1494740
P18	199464	393132	393132	159568	259080
P19	388928	952656	952652	567148	834308
P20	1598964	3227260	3227516	885456	1605568

Tabla 200. Memoria total utilizada – Termes – Evaluación

9.3.3 *Hiking*

Tiempo creación PDBs (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgHiking (Canonical PDB)*	*AlgHiking (Zero-One PDB)*
P01	0.0302866	0.000943414	0.00705679	0.00097664	0,00074142
P02	0.0712758	0.00645539	0.00866443	0.00179107	0,00136860
P03	0.106517	0.0101601	0.0218393	0.00515271	0,00629625
P04	0.120814	0.0771496	0.0488833	0.00784919	0,02167011
P05	0.167222	0.119798	0.141992	0.00852891	0,01819993
P06	0.0980582	0.312895	0.314841	0.00886705	0,01442009
P07	0.100794	0.203158	0.209333	0.01482607	0,02711509
P08	0.141477	0.207896	0.224114	0.03597609	0,03400630
P09	0.197075	0.20817	0.230865	0.07922309	0,07674127
P10	0.243474	0.621571	0.670453	0.18961792	0,25188301
P11	0.303755	0.316681	0.450755	0.33960904	0,42677794
P12	0.17687	0.334628	0.417574	0.02973976	0,06760469
P13	0.198221	0.474653	0.532713	0.04637520	0,04644781
P14	0.271541	0.364617	0.404445	0.10534140	0,18210402
P15	0.352761	1.19161	1.38873	0.22397822	0,25435800
P16	0.161653	0.871883	0.823142	0.01082238	0,00852812
P17	0.157662	0.418876	0.488967	0.02457076	0,08490649

P18	0.231523	0.437852	0.455687	0.08592869	0,13650711
P19	0.283346	0.402768	0.457826	0.13268088	0,14850993
P20	0.515094	1.38856	1.28121	0.28911000	0,31687492

Tabla 201. Tiempo de creación de PDBs – Hiking – Evaluación

Tiempo poda y creación conjuntos aditivos (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgHiking (Canonical PDB)*	*AlgHiking (Zero-One PDB)*
P01	1.4625e-05	1.4433e-05	-	1.4733e-05	-
P02	1.4005e-05	2.8921e-05	-	1.3966e-05	-
P03	1.5161e-05	1.6536e-05	-	2.8312e-05	-
P04	1.4769e-05	2.7749e-05	-	2.6524e-05	-
P05	1.6793e-05	2.5017e-05	-	1.4049e-05	-
P06	2.2802e-05	3.2139e-05	-	4.2646e-05	-
P07	2.1398e-05	1.5986e-05	-	2.3507e-05	-
P08	2.2204e-05	3.1618e-05	-	2.3009e-05	-
P09	2.3431e-05	2.4889e-05	-	4.401e-05	-
P10	2.0724e-05	2.4894e-05	-	2.6667e-05	-
P11	2.3724e-05	2.352e-05	-	3.3386e-05	-
P12	2.017e-05	3.1773e-05	-	3.7151e-05	-
P13	1.9802e-05	2.5147e-05	-	2.2551e-05	-
P14	2.008e-05	2.4169e-05	-	3.353e-05	-

P15	2.0399e-05	2.4196e-05	-	3.7757e-05	-
P16	2.0445e-05	2.4613e-05	-	2.2005e-05	-
P17	2.0506e-05	2.4425e-05	-	3.3423e-05	-
P18	2.0338e-05	2.7835e-05	-	3.4587e-05	-
P19	2.213e-05	2.4187e-05	-	3.3846e-05	-
P20	2.6382e-05	1.5834e-05	-	3.4617e-05	-

Tabla 202. Tiempo poda y creación conjuntos aditivos – Hiking – Evaluación

Tiempo búsqueda (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgHiking (Canonical PDB)*	*AlgHiking (Zero-One PDB)*
P01	0.00076818	0.000811107	0.00350587	0.000790129	0.000779544
P02	0.00269352	0.00236588	0.00223882	0.0027499	0.00278669
P03	0.0147891	0.00197827	0.00318774	0.0232137	0.0233803
P04	0.146013	0.00315818	0.00317404	0.166074	0.212063
P05	0.343865	0.00397356	0.00399744	0.352403	0.395413
P06	0.0979858	0.00120724	0.00118555	0.0168288	0.0408324
P07	3.58218	2.52226	2.21045	0.169827	0.499556
P08	52.6621	36.8623	35.1415	1.43896	6.37217
P09	450.416	395.27	381.492	11.207	60.2366
P10	-	-	-	52.7792	334.498
P11	-	-	-	243.351	-

P12	14.4779	3.36013	3.25705	0.535736	1.5054
P13	269.142	224.858	215.606	6.10236	22.3243
P14	-	-	-	41.8914	187.039
P15	-	-	-	274.184	-
P16	0.679197	0.166396	0.166744	0.0442651	0.102099
P17	48.4345	9.00848	8.81983	1.1854	3.13029
P18	-	171.632	165.62	14.7317	47.0457
P19	-	-	-	146.386	-
P20	-	-	-	-	-

Tabla 203. Tiempo búsqueda – Hiking – Evaluación

Memoria total utilizada (KB)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgHiking (Canonical PDB)***	***AlgHiking (Zero-One PDB)***
P01	4492	4492	4492	4492	4492
P02	4492	4492	4492	4492	4492
P03	4752	4664	4664	4628	4628
P04	6240	6264	6264	6204	6196
P05	8148	8172	8176	8212	8212
P06	5536	6704	6708	4760	5024
P07	32148	30132	30156	7912	11484
P08	375440	379700	379616	29868	102576

P09	2884332	2905260	2905260	119288	744928
P10	-	-	-	473320	3067284
P11	-	-	-	2947008	-
P12	98956	33404	33500	12440	29068
P13	1675036	1562176	1562296	109516	256352
P14	-	-	-	460568	1876896
P15	-	-	-	3178112	-
P16	10976	10088	10092	5560	6340
P17	237440	100284	100340	21160	54168
P18	-	1083348	1083336	212552	524060
P19	-	-	-	1598396	-
P20	-	-	-	-	-

Tabla 204. Memoria total utilizada – Hiking – Evaluación

9.3.4 *Spider*

Tiempo creación PDBs (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSpider (Canonical PDB)*	*AlgSpider (Zero-One PDB)*
P01	3.83517	0.845935	0.958085	0.06697550	0,02499188
P02	8.75651	0.658853	0.732229	0.11880961	0,11746692
P03	15.9886	1.80162	2.001	0.35874495	0,33740703

P04	45.262	0.614497	0.364803	0.39930907	0,48052396
P05	46.0363	1.11381	0.77315	0.98704095	0,98574603
P06	85.4077	2.13395	1.53373	1.65879197	1,95614899
P07	3.04203	0.878562	0.9321	0.06305919	0,14489795
P08	4.44174	0.64478	0.776919	0.10656921	0,17856807
P09	14.1432	1.52269	1.63317	0.35036401	0,29292801
P10	132.326	0.357406	0.540732	0.38237717	0,41995090
P11	46.5612	0.811487	1.2137	0.89051884	0,91134414
P12	103.32	1.48316	2.09954	1.73945802	1,78382415
P13	478.321	2.52519	3.19675	2.12731504	2,23845997
P14	30.0897	0.615159	0.727489	0.06935673	0,06456932
P15	4.94848	0.627326	0.711365	0.13000006	0,16405814
P16	13.417	1.69283	1.88842	0.27530311	0,29849803
P17	18.5708	0.347056	0.540313	0.37557295	0,46399520
P18	45.2934	0.754535	1.13667	0.86963609	1,00088490
P19	89.4054	1.50482	2.07136	1.67595408	1,80340110
P20	115.432	2.3517	3.06185	2.06112902	2,18108609

Tabla 205. Tiempo de creación de PDBs – Spider – Evaluación

Tiempo poda y creación conjuntos aditivos (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSpider (Canonical PDB)*	*AlgSpider (Zero-One PDB)*
P01	0.000250002	8.038e-05	-	0.00343305	-
P02	0.000347986	0.000150833	-	0.00588104	-
P03	0.000470333	0.000143141	-	0.0217941	-
P04	0.000426996	0.000284676	-	0.0377007	-
P05	0.000594858	0.000309921	-	0.0208604	-
P06	0.000440028	0.00037315	-	0.0335111	-
P07	0.000189717	8.8543e-05	-	0.00103279	-
P08	0.000339253	0.00011772	-	0.00287435	-
P09	0.000325122	0.000285866	-	0.00367831	-
P10	0.00036544	0.000266981	-	0.0313585	-
P11	0.000479415	0.000309691	-	0.0698539	-
P12	0.000565493	0.000346195	-	0.0789613	-
P13	0.00046948	0.000407779	-	0.0495497	-
P14	0.000341072	0.000117943	-	0.00175342	-
P15	0.00028252	0.000119997	-	0.00288218	-
P16	0.000347726	0.00016657	-	0.01817	-
P17	0.00032308	0.000277775	-	0.0104088	-

P18	0.000478106	0.000313718	-	0.0502802	-
P19	0.000549221	0.000360155	-	0.0914225	-
P20	0.000617372	0.000396061	-	0.117316	-

Tabla 206. Tiempo poda y creación conjuntos aditivos – Spider – Evaluación

Tiempo búsqueda (s)	*iPDB*	*Canonical PDB (combo)*	*Zero-One PDB (combo)*	*AlgSpider (Canonical PDB)*	*AlgSpider (Zero-One PDB)*
P01	0.0236527	0.0566997	0.0649141	0.0506819	0.0238073
P02	0.0520457	0.0612246	0.0725504	0.128687	0.0499232
P03	0.424613	1.20571	27.4684	2.4278	0.483354
P04	0.442874	2.0714	42.5192	8.94524	1.53624
P05	6.40169	8.8058	-	19.757	6.39294
P06	-	-	-	-	-
P07	0.0191179	0.0236516	0.0260879	0.0252662	0.0220616
P08	0.0719355	0.212185	0.458393	0.136221	0.0835838
P09	0.685388	1.55192	22.2195	1.22043	0.727216
P10	0.386888	5.0457	201.608	13.7322	2.71129
P11	244.826	-	-	2289.6	281.81
P12	-	-	-	-	-
P13	440.856	-	-	-	-
P14	0.00722213	0.0619838	0.103786	0.0374925	0.0340325

P15	0.00628355	0.008457	0.012716	0.011412	0.0065704
P16	1.91498	7.54831	129.47	8.73577	2.82555
P17	9.37254	48.6508	-	25.6772	13.5849
P18	381.54	-	-	-	-
P19	-	-	-	-	-
P20	-	-	-	-	-

Tabla 207. Tiempo búsqueda – Spider – Evaluación

Memoria total utilizada (KB)	***iPDB***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgSpider (Canonical PDB)***	***AlgSpider (Zero-One PDB)***
P01	8572	13712	13712	5848	5848
P02	8624	17976	17976	6668	6668
P03	12916	21836	196068	11980	12232
P04	14360	24856	265180	18548	18748
P05	57532	74060	-	45260	48116
P06	-	-	-	-	-
P07	8408	12404	12404	5832	5832
P08	8384	14944	14944	6988	6984
P09	18956	24600	180612	13784	14092
P10	14644	43624	1485512	25608	27516
P11	1723672	-	-	1714800	1982228

P12	-	-	-	-	-
P13	3125868	-	-	-	-
P14	7572	12804	12804	5844	6024
P15	8384	15992	15992	6572	6572
P16	26628	94640	1280184	22412	35796
P17	106592	487000	-	71276	142292
P18	3696324	-	-	-	-
P19	-	-	-	-	-
P20	-	-	-	-	-

Tabla 208. Memoria total utilizada – Spider – Evaluación

9.3.5 *Tetris*

Tiempo creación PDBs (s)	***iPDB (max time = 60)***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTetris (Canonical PDB)***	***AlgTetris (Zero-One PDB)***
P01	113.487	7.7279	7.64804	8.04264188	8,24227599
P02	125.244	9.84795	10.2166	9.99716210	10,05523499
P03	76.6146	8.28203	8.2193	15.15274107	15,83564320
P04	77.5288	3.42782	3.49033	0.41983391	0,41176299
P05	62.3322	9.49419	9.73246	9.68956204	9,94938695
P06	81.2416	9.90559	9.92731	10.38388288	10,55674886

P07	60.0664	8.20779	8.19222	14.94758393	15,78612493
P08	63.8837	3.1981	3.3405	0.39196002	0,38731414
P09	98.9346	7.55985	7.61143	8.29564290	8,30628802
P10	65.1827	14.1649	14.4939	14.90916089	14,91035498
P11	72.4841	14.5518	14.9542	29.39397521	28,60100699
P12	60.0246	7.51444	7.68071	7.98877503	8,00411088
P13	60.3391	12.468	12.6782	12.05480498	12,16868398
P14	60.1569	12.2832	12.7976	23.87791097	25,48297020
P15	88.2097	7.40555	7.69072	7.98940586	7,80638992
P16	63.5459	14.4731	14.7271	15.44581104	14,14449208
P17	69.658	14.1116	14.4183	27.69039498	27,57060792

Tabla 209. Tiempo de creación de PDBs – Tetris – Evaluación

Tiempo poda y creación conjuntos aditivos (s)	***iPDB (max time* = 60)**	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTetris (Canonical PDB)***	***AlgTetris (Zero-One PDB)***
P01	4.9601e-05	3.6274e-05	-	6.8026e-05	-
P02	0.000552493	2.7934e-05	-	4.8093e-05	-
P03	0.000734622	4.2057e-05	-	6.4774e-05	-
P04	0.000142598	3.9976e-05	-	3.188e-05	-
P05	0.000497644	2.6961e-05	-	4.7672e-05	-
P06	0.000448127	2.7663e-05	-	4.658e-05	-

P07	0.000569253	3.9847e-05	-	7.3248e-05	-
P08	5.3343e-05	2.8944e-05	-	3.2098e-05	-
P09	4.7519e-05	3.0866e-05	-	4.9637e-05	-
P10	0.00013402	3.523e-05	-	6.2519e-05	-
P11	0.000276429	4.2914e-05	-	6.4596e-05	-
P12	0.00104448	2.9212e-05	-	5.0483e-05	-
P13	0.000115074	4.2551e-05	-	5.285e-05	-
P14	0.000550668	4.7094e-05	-	5.5406e-05	-
P15	3.9713e-05	3.1231e-05	-	4.999e-05	-
P16	0.00012882	4.1004e-05	-	9.0875e-05	-
P17	0.000277095	4.5029e-05	-	5.9539e-05	-

Tabla 210. Tiempo poda y creación conjuntos aditivos – Tetris – Evaluación

Tiempo búsqueda (s)	***iPDB (max time = 60)***	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTetris (Canonical PDB)***	***AlgTetris (Zero-One PDB)***
P01	3.10804	0.152322	0.146752	0.0383318	0.0391502
P02	2.80441	0.257891	0.258964	0.138235	0.138191
P03	-	-	-	-	-
P04	0.00068006	0.000563383	0.000560171	0.000506587	0.000542569
P05	1.11978	0.071907	0.0719823	0.0326089	0.0437963
P06	-	27.5233	27.381	13.6179	13.6032

P07	-	-	-	-	-
P08	0.00077357	0.000717048	0.000811841	0.000730823	0.000762509
P09	0.640852	0.0148014	0.0147438	0.00144727	0.00159573
P10	59.4728	5.37691	5.42231	3.77796	3.75284
P11	-	-	-	-	-
P12	5.70829	0.0689221	0.0686115	0.0484465	0.0493491
P13	-	312.306	312.168	199.72	214.162
P14	-	-	-	-	-
P15	0.0312283	0.00464395	0.00478285	0.00326475	0.00333605
P16	-	-	-	-	-
P17	-	-	-	260.602	252.775

Tabla 211. Tiempo búsqueda – Tetris – Evaluación

Memoria total utilizada (KB)	***iPDB (max time* = 60)**	***Canonical PDB (combo)***	***Zero-One PDB (combo)***	***AlgTetris (Canonical PDB)***	***AlgTetris (Zero-One PDB)***
P01	56128	11928	11928	11644	11664
P02	62360	15340	15372	14908	14864
P03	-	-	-	-	-
P04	60224	10108	10108	6280	6260
P05	27656	13896	13924	13608	13632
P06	-	400356	400468	210088	210088

P07	-	-	-	-	-
P08	37240	10104	10104	6232	6284
P09	50488	11684	11816	11568	11588
P10	559840	82408	82364	59752	59736
P11	-	-	-	-	-
P12	100192	11928	11928	11644	11664
P13	-	4073612	4073080	2802084	2802148
P14	-	-	-	-	-
P15	46976	11760	11760	11492	11512
P16	-	-	-	-	-
P17	-	-	-	2681092	2681156

Tabla 212. Memoria total utilizada – Tetris – Evaluación

9.3.6 Independiente del dominio

Tiempo creación PDBs (s)	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	897.011	5.69984393	200.58652195358277
P02	279.025	9.78233408	270.1129011116028
P03	346.773	11.42436882	337.13423403739927
P04	84.4693	0.11052613	366.1104518699646
P05	-	0.59178407	292.00137694358824

P06	446.251	13.07605392	447.84956792831423
P07	300.992	17.23160521	302.705660987854
P08	412.976	24.31779686	415.22257393074034
P09	912.14	0.33561602	406.33400299072264
P10	430.822	6.80652390	284.3329829940796
P11	641.545	15.48361493	644.4058170833588
P12	517.429	23.69320990	517.0690430583954
P13	691.745	31.17903194	695.5898231372834
P14	452.627	48.48496502	456.730487953186
P15	954.81	2.60191696	376.0533479423523
P16	705.208	28.10543607	709.3374349422455
P17	994.748	39.39794115	999.2849391479492
P18	679.166	56.25695809	684.2069290103913
P19	896.385	84.95284791	900.086780128479
P20	1173.1	9.92919088	772.953080953598

Tabla 213. Tiempo de creación de PDBs – Snake – DI

Tiempo creación PDBs (s)	***iPDB***	***AlgTermes (Canonical PDB)***	***iPDB modificado***
P01	3.56702	0.00926041	3.3842178691864016
P02	3.49414	0.01979477	3.3909089012145994

P03	4.28782	0.02051925	4.398945952758789
P04	4.41132	0.14733695	4.303046974334717
P05	3.08709	0.01958222	3.1689699996948244
P06	3.07474	0.42538806	3.2381199031066896
P07	2.35627	0.01711157	1.5246399756622315
P08	1.53309	6.62498118	1.6399988240051269
P09	2.57898	0.04013943	2.6482820610046387
P10	2.60056	0.40316097	2.619082963409424
P11	4.92451	0.71341210	3.4844209912872315
P12	3.39212	1.00851204	3.5264370392608644
P13	4.47534	1.13943313	4.5143949462890625
P14	4.33649	0.25544107	4.410401928329468
P15	2.98447	3.18947212	3.049577102584839
P16	1.7994	43.21525800	1.759537082977295
P17	3.73993	8.51733798	3.8034360009765624
P18	3.02609	5.35232609	3.0328160931396484
P19	4.68182	41.68896885	4.842485972366333
P20	2.68327	7.17378813	2.7051709474945067

Tabla 214. Tiempo de creación de PDBs – Termes – DI

Tiempo creación PDBs (s)	*iPDB*	*AlgHiking (Canonical PDB)*	*iPDB modificado*
P01	0.0302866	0.00097664	0.03398433495788574
P02	0.0712758	0.00179107	0.07095115054016113
P03	0.106517	0.00515271	0.07635163038024903
P04	0.120814	0.00784919	0.1354140118713379
P05	0.167222	0.00852891	0.19762911387634277
P06	0.0980582	0.00886705	0.12360596047973633
P07	0.100794	0.01482607	0.1785121866455078
P08	0.141477	0.03597609	0.18200993731689452
P09	0.197075	0.07922309	0.22246298834228515
P10	0.243474	0.18961792	0.2940129458465576
P11	0.303755	0.33960904	0.45656894635009765
P12	0.17687	0.02973976	0.15121009106445313
P13	0.198221	0.04637520	0.20630187071228026
P14	0.271541	0.10534140	0.4062528581390381
P15	0.352761	0.22397822	0.5044472057800293
P16	0.161653	0.01082238	0.12714499169921875
P17	0.157662	0.02457076	0.17870416592407226
P18	0.231523	0.08592869	0.3198419362487793

P19	0.283346	0.13268088	0.4403289240875244
P20	0.515094	0.28911000	0.6032310983886718

Tabla 215. Tiempo de creación de PDBs – Hiking – DI

Tiempo creación PDBs (s)	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	3.83517	0,02499188	3.8557289346313475
P02	8.75651	0,11746692	8.762547993011475
P03	15.9886	0,33740703	16.14733108100891
P04	45.262	0,48052396	13.615681076049805
P05	46.0363	0,98574603	46.29413982887268
P06	85.4077	1,95614899	86.38935916938782
P07	3.04203	0,14489795	3.1564799687194824
P08	4.44174	0,17856807	4.559953965530395
P09	14.1432	0,29292801	14.258488869476318
P10	132.326	0,41995090	24.18710107002258
P11	46.5612	0,91134414	46.92582896270752
P12	103.32	1,78382415	103.81496898841858
P13	478.321	2,23845997	119.60223512268067
P14	30.0897	0,06456932	2.899185022277832
P15	4.94848	0,16405814	5.127153017883301

P16	13.417	0,29849803	13.796008097076417
P17	18.5708	0,46399520	18.51462001991272
P18	45.2934	1,00088490	44.59261591453552
P19	89.4054	1,80340110	89.94355696868897
P20	115.432	2,18108609	116.52934693527222

Tabla 216. Tiempo de creación de PDBs – Spider – DI

Tiempo creación PDBs (s)	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*
P01	113.487	8.04264188	113.7820539264679
P02	125.244	9.99716210	85.62067000350952
P03	76.6146	15.15274107	76.93079296150208
P04	77.5288	0.41983391	73.30265084571839
P05	62.3322	9.68956204	62.6400490032196
P06	81.2416	10.38388288	81.61782189788818
P07	60.0664	14.94758393	60.38673792991638
P08	63.8837	0.39196002	107.7896930217743
P09	98.9346	8.29564290	99.08613203735352
P10	65.1827	14.90916089	65.79421297264099
P11	72.4841	29.39397521	75.12112784385681
P12	60.0246	7.98877503	111.28003100967408

P13	60.3391	12.05480498	61.306338082504276
P14	60.1569	23.87791097	60.64218387374878
P15	88.2097	7.98940586	88.59656788406372
P16	63.5459	15.44581104	64.23544989395141
P17	69.658	27.69039498	73.03094078254699

Tabla 217. Tiempo de creación de PDBs – Tetris – DI

Tiempo poda y creación conjuntos aditivos (s)	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	0.000114591	7.9294e-05	0.000147165
P02	0.000204172	0.000114895	0.00016569
P03	0.000104696	8.2989e-05	9.7128e-05
P04	4.6969e-05	5.1991e-05	6.6716e-05
P05	-	6.5925e-05	0.000129984
P06	8.1162e-05	9.3987e-05	8.1307e-05
P07	7.7994e-05	0.000127157	8.5885e-05
P08	0.000101461	0.000116475	0.000104701
P09	6.336e-05	5.3749e-05	6.0745e-05
P10	0.000241553	0.000111777	0.00019211
P11	8.4443e-05	0.000102429	8.7209e-05
P12	0.000100337	8.9804e-05	0.000101991

P13	0.000125804	9.1854e-05	0.00012762
P14	0.000130829	0.000124142	0.000140669
P15	7.7087e-05	4.7566e-05	7.3832e-05
P16	8.8873e-05	0.000121876	9.0906e-05
P17	0.000104036	0.000116227	0.000115329
P18	0.000130395	8.8725e-05	0.000130126
P19	0.000149052	9.7827e-05	0.000156741
P20	8.7738e-05	0.000101114	8.1909e-05

Tabla 218. Tiempo poda y creación conjuntos aditivos – Snake – DI

Tiempo poda y creación conjuntos aditivos (s)	***iPDB***	***AlgTermes (Canonical PDB)***	***iPDB modificado***
P01	4.9804e-05	5.8889e-05	4.8659e-05
P02	6.1458e-05	6.7321e-05	4.6794e-05
P03	7.8074e-05	4.7878e-05	5.7574e-05
P04	6.7799e-05	5.5715e-05	4.6885e-05
P05	5.826e-05	5.1274e-05	5.7585e-05
P06	4.5084e-05	5.4754e-05	4.5634e-05
P07	5.0743e-05	0.000137399	5.5144e-05
P08	5.7357e-05	5.6914e-05	5.6333e-05
P09	4.2588e-05	0.00013703	4.3479e-05

P10	5.026e-05	5.3525e-05	5.0799e-05
P11	5.2188e-05	5.2686e-05	5.1982e-05
P12	4.4521e-05	7.4213e-05	5.3451e-05
P13	4.2446e-05	0.000111646	4.5799e-05
P14	4.0861e-05	5.7211e-05	4.2246e-05
P15	4.1828e-05	5.0921e-05	5.0763e-05
P16	3.7237e-05	8.0752e-05	3.468e-05
P17	4.6544e-05	0.000139167	4.1766e-05
P18	4.7081e-05	0.000106998	4.6708e-05
P19	3.9261e-05	0.000125596	3.798e-05
P20	4.0351e-05	6.0391e-05	4.7611e-05

Tabla 219. Tiempo poda y creación conjuntos aditivos – Termes – DI

Tiempo poda y creación conjuntos aditivos (s)	***iPDB***	***AlgHiking (Canonical PDB)***	***iPDB modificado***
P01	1.4625e-05	1.4733e-05	1.4989e-05
P02	1.4005e-05	1.3966e-05	1.5584e-05
P03	1.5161e-05	2.8312e-05	1.7132e-05
P04	1.4769e-05	2.6524e-05	1.5815e-05
P05	1.6793e-05	1.4049e-05	1.6753e-05
P06	2.2802e-05	4.2646e-05	2.2646e-05

P07	2.1398e-05	2.3507e-05	2.3439e-05
P08	2.2204e-05	2.3009e-05	2.2567e-05
P09	2.3431e-05	4.401e-05	2.498e-05
P10	2.0724e-05	2.6667e-05	2.1869e-05
P11	2.3724e-05	3.3386e-05	2.3739e-05
P12	2.017e-05	3.7151e-05	2.392e-05
P13	1.9802e-05	2.2551e-05	2.2513e-05
P14	2.008e-05	3.353e-05	2.3095e-05
P15	2.0399e-05	3.7757e-05	2.3928e-05
P16	2.0445e-05	2.2005e-05	2.2524e-05
P17	2.0506e-05	3.3423e-05	2.2868e-05
P18	2.0338e-05	3.4587e-05	2.2456e-05
P19	2.213e-05	3.3846e-05	2.2723e-05
P20	2.6382e-05	3.4617e-05	2.329e-05

Tabla 220. Tiempo poda y creación conjuntos aditivos – Hiking – DI

Tiempo poda y creación conjuntos aditivos (s)	***iPDB***	***AlgSpider (Zero-One PDB)***	***iPDB modificado***
P01	0.000250002	-	0.000250705
P02	0.000347986	-	0.000367694
P03	0.000470333	-	0.000430231

P04	0.000426996	-	0.000517114
P05	0.000594858	-	0.000601254
P06	0.000440028	-	0.00044579
P07	0.000189717	-	0.000194269
P08	0.000339253	-	0.000289735
P09	0.000325122	-	0.000451869
P10	0.00036544	-	0.000428068
P11	0.000479415	-	0.000479421
P12	0.000565493	-	0.000629439
P13	0.00046948	-	0.000498914
P14	0.000341072	-	0.000210806
P15	0.00028252	-	0.00027478
P16	0.000347726	-	0.000337928
P17	0.00032308	-	0.000330375
P18	0.000478106	-	0.00048229
P19	0.000549221	-	0.00061003
P20	0.000617372	-	0.000624491

Tabla 221. Tiempo poda y creación conjuntos aditivos – Spider – DI

Tiempo poda y creación conjuntos aditivos (s)	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*
P01	4.9601e-05	6.8026e-05	7.7318e-05
P02	0.000552493	4.8093e-05	0.000979779
P03	0.000734622	6.4774e-05	0.00071849
P04	0.000142598	3.188e-05	0.000133742
P05	0.000497644	4.7672e-05	0.000662084
P06	0.000448127	4.658e-05	0.000530383
P07	0.000569253	7.3248e-05	0.000482474
P08	5.3343e-05	3.2098e-05	0.000108145
P09	4.7519e-05	4.9637e-05	4.7822e-05
P10	0.00013402	6.2519e-05	0.000138365
P11	0.000276429	6.4596e-05	0.000287116
P12	0.00104448	5.0483e-05	0.00134435
P13	0.000115074	5.285e-05	0.000108184
P14	0.000550668	5.5406e-05	0.0005529
P15	3.9713e-05	4.999e-05	3.6188e-05
P16	0.00012882	9.0875e-05	0.000116715
P17	0.000277095	5.9539e-05	0.000290603

Tabla 222. Tiempo poda y creación conjuntos aditivos – Tetris – DI

Tiempo búsqueda (s)	*iPDB*	*AlgSnake (Canonical PDB)*	*iPDB modificado*
P01	0.00180985	0.00173066	0.00167023
P02	0.0558545	0.0710127	0.0547254
P03	14.5863	4.36722	14.2928
P04	0.0007224	0.000887084	0.000814104
P05	-	0.00104429	0.00102938
P06	0.010764	0.0165334	0.00528399
P07	212.179	37.3199	214.422
P08	-	634.851	-
P09	0.00122594	0.00116513	0.00119228
P10	0.00175552	0.00196234	0.00176221
P11	0.0067074	0.00694257	0.00671335
P12	149.223	27.4568	150.608
P13	-	-	-
P14	-	-	-
P15	0.00179786	0.00177367	0.0017742
P16	4.0889	0.799301	4.16101
P17	-	595.17	-
P18	-	-	-

P19	-	-	-
P20	0.00192742	0.00193004	0.00199719

Tabla 223. Tiempo búsqueda – Snake – DI

Tiempo búsqueda (s)	*iPDB*	*AlgTermes (Canonical PDB)*	*iPDB modificado*
P01	0.0215248	0.102681	0.0218034
P02	0.0749325	0.604941	0.0735987
P03	0.564836	6.06744	0.541339
P04	2.9279	5.41609	2.893
P05	268.81	-	268.66
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	0.820775	0.51604	2.59704
P12	0.202634	0.269315	0.207377
P13	53.6902	52.3036	53.8074
P14	7.35762	35.357	7.41328
P15	-	-	-

P16	-	-	-
P17	78.0712	91.8268	78.2927
P18	12.5837	9.2814	12.5784
P19	22.3321	16.312	22.4909
P20	124.186	66.6737	124.201

Tabla 224. Tiempo búsqueda – Termes – DI

Tiempo búsqueda (s)	*iPDB*	*AlgHiking (Canonical PDB)*	*iPDB modificado*
P01	0.00076818	0.000790129	0.000777436
P02	0.00269352	0.0027499	0.00291896
P03	0.0147891	0.0232137	0.0139824
P04	0.146013	0.166074	0.143948
P05	0.343865	0.352403	0.352971
P06	0.0979858	0.0168288	0.0959745
P07	3.58218	0.169827	3.77297
P08	52.6621	1.43896	53.5507
P09	450.416	11.207	457.581
P10	-	52.7792	-
P11	-	243.351	-
P12	14.4779	0.535736	14.5803

P13	269.142	6.10236	274.447
P14	-	41.8914	-
P15	-	274.184	-
P16	0.679197	0.0442651	0.63478
P17	48.4345	1.1854	46.8745
P18	-	14.7317	-
P19	-	146.386	-
P20	-	-	-

Tabla 225. Tiempo búsqueda – Hiking – DI

Tiempo búsqueda (s)	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	0.0236527	0.0238073	0.0241218
P02	0.0520457	0.0499232	0.0520278
P03	0.424613	0.483354	0.433041
P04	0.442874	1.53624	1.44374
P05	6.40169	6.39294	6.50081
P06	-	-	-
P07	0.0191179	0.0220616	0.0192654
P08	0.0719355	0.0835838	0.0733892
P09	0.685388	0.727216	0.68998

P10	0.386888	2.71129	2.46939
P11	244.826	281.81	252.365
P12	-	-	-
P13	440.856	-	-
P14	0.00722213	0.0340325	0.0259055
P15	0.00628355	0.0065704	0.00628808
P16	1.91498	2.82555	1.94095
P17	9.37254	13.5849	9.33921
P18	381.54	-	384.203
P19	-	-	-
P20	-	-	-

Tabla 226. Tiempo búsqueda – Spider – DI

Tiempo búsqueda (s)	*iPDB (max time = 60)*	*AlgTetris (Canonical PDB)*	*iPDB modificado (max time = 60)*
P01	3.10804	0.0383318	3.1204
P02	2.80441	0.138235	2.07973
P03	-	-	-
P04	0.00068006	0.000506587	0.000689808
P05	1.11978	0.0326089	1.12139
P06	-	13.6179	-

P07	-	-	-
P08	0.00077357	0.000730823	0.000748452
P09	0.640852	0.00144727	0.775056
P10	59.4728	3.77796	59.7743
P11	-	-	-
P12	5.70829	0.0484465	3.75583
P13	-	199.72	-
P14	-	-	-
P15	0.0312283	0.00326475	0.031251
P16	-	-	-
P17	-	260.602	-

Tabla 227. Tiempo búsqueda – Tetris – DI

Memoria total utilizada (KB)	***iPDB***	***AlgSnake (Canonical PDB)***	***iPDB modificado***
P01	3966840	49484	995384
P02	1375808	70800	1375808
P03	1692756	101756	1692760
P04	614976	7412	1158716
P05	-	10752	1202248
P06	2008968	84240	2008968

P07	1282872	338372	1282872
P08	-	2898312	-
P09	3456680	9988	1486580
P10	1916044	49564	1291976
P11	2469328	90648	2469328
P12	1869348	275728	1869340
P13	-	-	-
P14	-	-	-
P15	3614664	22448	1469752
P16	2264672	132312	2264664
P17	-	3007224	-
P18	-	-	-
P19	-	-	-
P20	3806524	56764	2555996

Tabla 228. Memoria total utilizada – Snake – DI

Memoria total utilizada (KB)	***iPDB***	***AlgTermes (Canonical PDB)***	***iPDB modificado***
P01	49240	6104	49248
P02	49080	11744	49080
P03	61760	102988	61760

P04	63816	103692	63660
P05	3288164	-	3288348
P06	-	-	-
P07	-	-	-
P08	-	-	-
P09	-	-	-
P10	-	-	-
P11	67124	21188	52152
P12	49072	16796	49120
P13	766876	775904	766808
P14	114188	449900	114196
P15	-	-	-
P16	-	-	-
P17	847824	929328	847728
P18	199464	159568	199372
P19	388928	567148	388944
P20	1598964	885456	1598960

Tabla 229. Memoria total utilizada – Termes – DI

Memoria total utilizada (KB)	*iPDB*	*AlgHiking (Canonical PDB)*	*iPDB modificado*
P01	4492	4492	4492
P02	4492	4492	4492
P03	4752	4628	4752
P04	6240	6204	6240
P05	8148	8212	8148
P06	5536	4760	5536
P07	32148	7912	32160
P08	375440	29868	375440
P09	2884332	119288	2884928
P10	-	473320	-
P11	-	2947008	-
P12	98956	12440	98924
P13	1675036	109516	1675028
P14	-	460568	-
P15	-	3178112	-
P16	10976	5560	10952
P17	237440	21160	237380
P18	-	212552	-

P19	-	1598396	-
P20	-	-	-

Tabla 230. Memoria total utilizada – Hiking – DI

Memoria total utilizada (KB)	*iPDB*	*AlgSpider (Zero-One PDB)*	*iPDB modificado*
P01	8572	5848	8576
P02	8624	6668	8620
P03	12916	12232	12988
P04	14360	18748	20028
P05	57532	48116	57548
P06	-	-	-
P07	8408	5832	8408
P08	8384	6984	8384
P09	18956	14092	18932
P10	14644	27516	26952
P11	1723672	1982228	1723652
P12	-	-	-
P13	3125868	-	-
P14	7572	6024	11092
P15	8384	6572	8384

P16	26628	35796	26652
P17	106592	142292	106584
P18	3696324	-	3695868
P19	-	-	-
P20	-	-	-

Tabla 231. Memoria total utilizada – Spider – DI

Memoria total utilizada (KB)	***iPDB (max time = 60)***	***AlgTetris (Canonical PDB)***	***iPDB modificado (max time = 60)***
P01	56128	11644	56132
P02	62360	14908	33420
P03	-	-	-
P04	60224	6280	57076
P05	27656	13608	27656
P06	-	210088	-
P07	-	-	-
P08	37240	6232	55316
P09	50488	11568	50488
P10	559840	59752	559972
P11	-	-	-
P12	100192	11644	61280

P13	-	2802084	-
P14	-	-	-
P15	46976	11492	46980
P16	-	-	-
P17	-	2681092	-

Tabla 232. Memoria total utilizada – Tetris – DI

SECCIÓN 10. REFERENCIAS

Bonet, B., Patrik, H., Botea, A., Helmert, M., & Koening, S. (2007). Domain-Independent Construction of Pattern Database Heuristics for. *AAAI'07 Proceedings of the 22nd national conference on Artificial intelligence - Volume 2*, 1007-1012 .

Culberson, J. C., & Schaeffer, J. (1998). Pattern databases. *Computational Intelligence*, 318-334.

Edelkamp, S. (2002). Symbolic Pattern Databases in Heuristic Search Planning. *ICAPS 2005 - Proceedings of the 15th International Conference on Automated Planning and Scheduling*, 51-60.

Hart, P., Nilsson, N., & Raphael, B. (1968). Formal Basis for the Heuristic Determination of Minimum Cost Paths. *IEEE Transactions System Science and Cybernetics*, 100-107.

Helmert, M. (2018). *Fast Downward*. Obtenido de http://www.fast-downward.org/ [Consulta: 17 enero 2018]

Keller, T., Sanner, S., & Say, B. (2018). *ipc2018*. Obtenido de https://ipc2018.bitbucket.io/ [Consulta: 22 junio 2018]

Liaison, M. D. (2018). *icaps*. Obtenido de http://www.icaps-conference.org/index.php/Main/HomePage [Consulta: 12 enero 2018]

Maganto, Á. S. (2017). ESTUDIO DE TÉCNICAS SUPERVISADAS DE REDUCCIÓN DE DIMENSIONALIDAD PARA PROBLEMAS DE CLASIFICACIÓN. España: UC3M.

McDermott, D. (1998). PDDL The Planning Domain Definition Language.

NODDE. (2017). *Análisis del sistema de información.* Leganés.

Russell, S. J., & Norvig, P. (2003). Artificial Intelligence: A Modern Approach. *Upper Saddle River, New Jersey: Prentice Hall*, 111–114.

Sievers, S., Wehrle, M., & Helmert, M. (2014). Generalized Label Reduction for Merge-and-Shrink Heuristics. *AAAI'14 Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, 2358-2366.

Sievers, S., Wehrle, M., & Helmert, M. (2016). An Analysis of Merge Strategies for Merge-and-Shrink Heuristics. *ICAPS'16 Proceedings of the Twenty-Sixth International Conference on International Conference on Automated Planning and Scheduling*, 294-298.

Social, M. d. (2017). *Resolución de 30 de diciembre de 2016, de la Dirección General de Empleo, por la que se registra y publica el Convenio colectivo del sector de empresas de ingeniería y oficinas de estudios técnicos.*

Social, M. d. (2018). *Resolución de 22 de febrero de 2018, de la Dirección General de Empleo, por la que se registra y publica el XVII Convenio colectivo estatal de empresas de consultoría y estudios de mercado y de la opinión pública.*

UC3M. (2015). *ocw*. Obtenido de http://ocw.uc3m.es/historico/planificacion-automatica/material-de-clase [Consulta: 27 diciembre 2017]

UC3M. (2018). *Informe del Tutor del Trabajo Fin de Grado.*

UC3M. (2018). *Matriz de Evaluación de Trabajo Fin de Grado.*

UPV. (s.f.). Cómo escribir un Trabajo Fin de Grado.

wikis.fdi.ucm. (2016). Obtenido de http://wikis.fdi.ucm.es/ELP/TFG_-_A_qui%C3%A9n_pertenecen_los_derechos_de_propiedad_intelectual [Consulta: 30 enero 2018]